UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMATICA

TESIS DE GRADO

“BEEHIVE, UNA PLATAFORMA OPEN-SOURCE PARA EL INTERNET DE LAS COSAS”

PARA OPTAR AL TÍTULO DE LICENCIATURA EN INFORMÁTICA
MENCIÓN: INGENIERIA DE SISTEMAS INFORMÁTICOS

POSTULANTE: SERGIO GABRIEL GUILLEN MANTILLA
TUTORA METODOLOGICA: LIC. MENFY MORALES RÍOS
ASESORA: LIC. BRIGIDA ALEXANDRA CARVAJAL BLANCO

LA PAZ – BOLIVIA
2015

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMÁTICA

LA CARRERA DE INFORMÁTICA DE LA FACULTAD DE CIENCIAS PURAS Y NATURALES PERTENECIENTE A LA UNIVERSIDAD MAYOR DE SAN ANDRÉS AUTORIZA EL USO DE LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO SI LOS PROPÓSITOS SON ESTRICTAMENTE ACADÉMICOS.

LICENCIA DE USO

El usuario está autorizado a:

a) visualizar el documento mediante el uso de un ordenador o dispositivo móvil.
b) copiar, almacenar o imprimir si ha de ser de uso exclusivamente personal y privado.
c) copiar textualmente parte(s) de su contenido mencionando la fuente y/o haciendo la referencia correspondiente respetando normas de redacción e investigación.

El usuario no puede publicar, distribuir o realizar emisión o exhibición alguna de este material, sin la autorización correspondiente.

DEDICATORIA

A mis padres Ana y Luis que siempre me brindaron su apoyo y confianza en los buenos y malos momentos.

A mis abuelas Bertha (Q.E.P.D.) y Victoria de las que pude aprender valiosas enseñanzas a lo largo de mi vida.

A mis hermanos, tía y primas por brindarme el apoyo para poder convertir en realidad este trabajo.

A todos los amigos con los que compartí muchos momentos de alegría y tristeza.

AGRADECIMIENTOS

A mi docente y tutora metodológica Lic. Menfy Morales Ríos por su guía y esmero en la elaboración del presente trabajo, además de siempre haber brindado apoyo incondicional a toda la comunidad académica en la Carrera de Informática.

A mi docente asesora Lic. Brigida Alexandra Carvajal Blanco por los consejos, observaciones, correcciones y paciencia, sin los cuales el presente trabajo no sería lo que es.

Al Univ. Gustavo Alanoca por haberme brindado sus conocimientos en Electrónica los cuales fueron parte fundamental en la elaboración del presente trabajo.

A mi amigo Jhonatan Castro Rocabado con el que viví derrotas y triunfos, y cuyos consejos me permitieron ver desde otra perspectiva las diferentes situaciones.

A los amigos del Hacklab r00tHouse con los cuales tuve una retro alimentación útil para el presente trabajo.

RESUMEN

El Internet De Las Cosas es una propuesta que surge recientemente la cual plantea la conexión de cualquier dispositivo de hardware a Internet. Esta propuesta recibe más y más acogida dentro del mundo de la tecnología y negocios.

En la actualidad conectar dispositivos de hardware a través de Internet es totalmente viable gracias a los bajos costos en módulos de conexión a Internet para micro controladores y sistemas embebidos. Por otro lado, las diferentes tecnologías y protocolos de software que surgen continuamente, permiten a un desarrollador interactuar con dispositivos de hardware remotamente y con tiempos de respuesta reducidos.

Es importante recalcar que gran parte de los sistemas operativos y protocolos de comunicación para un dispositivo de hardware es software Open-Source.

El presente trabajo muestra la integración de diferentes componentes y el uso de diferentes protocolos de comunicación que resultan en el desarrollo de una plataforma Open-Source para el Internet De Las Cosas cuyo nombre es Beehive. Esta permite a fabricantes o entusiastas de la electrónica poder conectar diferentes dispositivos de hardware a través de Internet.

Las pruebas de hardware realizadas a esta propuesta de software muestran dos enfoques diferentes de hardware. La primera prueba orientada a la domótica, muestra como la plataforma Beehive puede monitorear y controlar remotamente diferentes sensores y actuadores dentro de un departamento. La segunda prueba se realiza con el sistema de frenado y la posición de un automóvil, un usuario tiene la posibilidad de controlar el sistema de frenado de un automóvil además de localizar la posición de un automóvil haciendo uso de la plataforma Beehive.

Palabras Clave: Internet de las Cosas, Open-Source, Micro controladores, Protocolos de Comunicación.

ABSTRACT

The Internet of Things is a new proposal that proposes the connection of any hardware device through Internet. This proposal gets more and more acceptance on the worlds of technology and business.

Nowadays connecting hardware devices through Internet is totally feasible because of the low costs on Internet connection modules available for micro controllers and embedded systems. On the other hand, the technologies and protocols that emerge continuously, allow developers interacting remotely with hardware devices with low response times.

It is important to mention that most operating systems and communication protocols for a hardware device are Open-Source Software.

This research shows the integration of different components, the use of different protocols that have as a result an Open-Source platform for the Internet of Things whose name is Beehive. This platform allows hardware makers to connect different hardware devices through Internet.

The hardware tests done to this platform shows two different points of view. The first test is oriented to domotics and shows how the Beehive platform can control remotely different sensors and actors inside a house. The second test controls the breaking system in a car and the position of it.

Keywords: Internet of Things, Open-Source, Micro controllers, Communication Protocols.

Índice de contenido

Índice de figuras

Índice de tablas

CAPÍTULO 1
MARCO REFERENCIAL

1.1. Introducción

En la actualidad es más simple y económico la comunicación entre personas en diferentes países o incluso continentes a través de Internet. Al ya tener el mundo la capacidad de mantener esta conexión, el siguiente paso que se desea es tener todo conectado (Personas y dispositivos de hardware).

El Internet De Las Cosas hace referencia a la interconexión entre dispositivos de hardware (televisores, sensores y otros) a través de Internet mediante el uso de diferentes protocolos, dominios y aplicaciones.

Según estudios, se estima que para el año 2020 existirán 26 billones de dispositivos conectados (excluyendo PCs, smartphones, tablets) es por eso que la cantidad de desarrolladores que será necesaria para poder trabajar con estos dispositivos y procesar la gran cantidad de datos que estos generen será alta.

Actualmente existen plataformas comerciales que ofrecen servicios para el Internet De Las Cosas entre diferentes dispositivos y sistemas como Xively, Jasper Technologies, Lockitron y NestLabs, permitiendo a fabricantes de hardware conectar sus dispositivos mediante estas plataformas brindando características como gestión de usuarios, gestión de permisos sobre dispositivos, librerías para micro controladores y la capacidad de ser extendida por otros desarrolladores.

Paralelamente a los productos comerciales para el Internet de las Cosas existe un conjunto de alternativas Open-Source, que si bien es más variado, los proyectos Open-Source relacionados al Internet de las Cosas que son de tipo plataforma no cuentan con las características principales que una plataforma comercial brinda.

Es por ello que el presente proyecto ofrece una alternativa Open-Source a dichas plataformas comerciales de tal forma que esta pueda funcionar como un servicio en la nube o alojada den-

tro de la red de una empresa que además tenga las principales características que brinda una plataforma comercial.

Esta plataforma plataforma permite a fabricantes de dispositivos de hardware y entusiastas de la electrónica, realizar la conexión de sus dispositivos a través de Internet sin la preocupación de tener que implementar todo un servicio de comunicación desde cero, al ser Open-Source esta plataforma puede adaptarse a las necesidades de una o más personas, además de poder crear instancias en los ambientes

El trabajo es desarrollado haciendo uso de la metodología de desarrollo Open Scrum. Los siete sprints realizados muestran paso a paso como esta plataforma es construida, desde la identificación de historias de usuario para el backlog inicial del producto, las tecnologías identificadas para la creación del presente hasta tener el producto final.

Las características principales de esta alternativa Open-Source son: canales de comunicación que permiten a los dispositivos conectarse a la plataforma, visualización en tiempo real de los datos generados por los dispositivos, gestión de usuarios, gestión de permisos sobre dispositivos, librerías para micro controladores y una API RESTful que permite a otros desarrolladores extender la plataforma.

Se realizaron dos pruebas a nivel de hardware desde dos puntos de vista diferentes relacionados a la domótica y a la mecánica.

La primera prueba es una instalación domótica en un supuesto departamento donde se visualizan y controlan diferentes sensores y actuadores a través de la plataforma Beehive. Haciendo uso de Raspberry Pi y Arduino como componentes principales de esta instalación es posible conectarse con la plataforma Beehive a través de los canales de comunicación que esta brinda.

La segunda prueba con un nivel de complejidad incrementado plantea un reto interesante. Esta prueba permite el control remoto de la posición de un automóvil además del control remoto del sistema de frenado de este. Es por ello que el dispositivo que se fabrica se conecta a Internet a través de la red 3G de celulares. De esta forma diferentes inconvenientes surgen como las caídas de señal y para la geolocalización bloqueo por el estado climatológico.

Los resultados que se obtienen (en tiempos de respuesta y concurrencia tras haber hecho las pruebas en un servidor con capacidades de hardware limitadas y un ancho de banda limitado) son alentadores ya que muestran que la elección de los protocolos de comunicación fueron los correctos.

1.2. Antecedentes

En la actualidad, existen empresas que ofrecen distintos tipos de productos de hardware y software para el Internet De Las Cosas. Muchas de estas empresas proveen plataformas de propósito general como otras de propósito específico.

De la misma forma, existen diferentes tipos de proyectos Open-Source relacionados al Internet De Las Cosas tanto hardware como software (plataformas, librerías para micro controladores, sistemas operativos empotrados y otros).

1.2.1. Antecedentes Comerciales

Jasper Technologies

Fundada el año 2005, Jasper Technologies, Inc[1]. es una empresa que provee software basado en la nube para el Internet De Las Cosas. A lo largo de su historia, esta viene ofreciendo servicios para diferentes industrias como: automotriz, domótica, agricultura, salud y otros. Jasper Technologies, Inc. trabaja con 11 fabricantes de automóviles, entre ellos, General Motors, Nissan, Ford, Tesla Motors. [JASPER: 2014]

La plataforma que Jasper provee, brinda a los clientes diagnósticos en tiempo real, empresas como Coca-Cola utilizan los servicios de Jasper para tener una conexión con las máquinas vendedoras, de tal forma que se pueda notificar inventario, estado de la máquina y otros en tiempo real y remotamente. [VCPOST: 2014]

Xively

Xively[2] (conocida anteriormente como Pachube, que posteriormente fue adquirida por

1 Jasper Technologies - www.jasper.com
2 Xively by LogMeIn – www.xively.com/

LogMeIn el año 2011) es una empresa que ofrece una plataforma para el Internet De Las Cosas con la capacidad de mantener conectados diferentes dispositivos a través de Internet. Entre los servicios basados en la nube que ofrece se tienen: servicios de datos, control de dispositivos a través de Internet y otros. Esta plataforma permite su extensión a través de interfaces de programación de aplicaciones (RESTful API). [XIVELY: 2011]

Para facilitar el uso de su plataforma a fabricantes o entusiastas, Xively provee librerías para diferentes micro controladores y sistemas operativos empotrados de diferentes arquitecturas que cuenten con una conexión a Internet. Gracias a estas librerías los dispositivos de hardware pueden disfrutar de todas las características que la plataforma de Xively provee a sus usuarios.

Además de permitir el uso a fabricantes o entusiastas, Xively también cuenta con una división la cual provee servicios a medida a empresas como: BioLabs, Turbid y Salesforce.

Lockitron

Lockitron[3] es una cerradura que puede ser controlada de forma remota y local. El dispositivo fue desarrollado por Apigy, una empresa establecida en California.

Estas cerraduras pueden ser bloqueadas o desbloqueadas vía web o por una aplicación móvil. Entre las características para el bloqueado y desbloqueado de estas cerraduras, un usuario puede asignar llaves virtuales a otros usuarios para que estos puedan tener control sobre la cerradura.

Entre las características de reusabilidad, Lockitron cuenta con una API RESTful la cual permite a otros desarrolladores el uso de su plataforma. [LOCKITRON: 2014]

Nest Labs

Comprada por Google en enero de 2014, Nest Labs[4] es una empresa de automatización ubicada en California. Esta empresa desarrolla termostatos y detectores de humo basados en sensores y módulos WiFi para la comunicación con su plataforma a través de Internet. La

3 Lockitron – www.lockitron.com/
4 Nest Labs – www.nest.com

plataforma que ofrece Nest Labs, permite a otros desarrolladores obtener información acerca de los dispositivos e interactuar con su plataforma a través de una API RESTful. [GOOGLE-NEST: 2014]

1.2.2. Antecedentes Open-Source

Paralelamente a los productos comerciales, surgen diferentes tipos de alternativas Open-Source que se encuentran relacionadas al Internet De Las Cosas.

Estos proyectos pueden catalogarse de la siguiente manera:

1. Sistemas Operativos Empotrados
2. Librerías para micro controladores
3. *Plataformas para el Internet De Las Cosas*

Al ser interés del presente trabajo, solo se tomarán en cuenta los proyectos Open-Source de tipo plataforma como antecedentes.

Meshblu

El proyecto Meshblu[5] provee principalmente canales de comunicación para los dispositivos de hardware. Al igual que otras plataformas comerciales, Meshblu provee interfaces de programación de aplicaciones RESTful, librerías para micro controladores y varios protocolos para los canales de comunicación (HTTP, MQTT, COAP y otros). A diferencia de otras plataformas, esta se base en su mayoría en el lenguaje JavaScript, desde la conexión con los canales de comunicación hasta las interfaces de programación.

Los canales de comunicación que esta provee son bidireccionales, es decir, un usuario puede enviar y recibir información de un dispositivo a través de su plataforma. [MESHBLU: 2014]

ThingSpeak

El proyecto ThingSpeak[6] provee canales de comunicación en una sola dirección, los

5 Proyecto Meshblu - https://github.com/octoblu/meshblu
6 Proyecto ThinkSpeak - thingspeak.com

dispositivos conectados a esta plataforma solo pueden enviar información de estado, sensores y otros solamente a través del protocolo HTTP. Una de las características importantes de esta plataforma es la facilidad que brinda para el trabajo con diferentes formatos (JSON, XML y CSV) para poder descargar la información que los dispositivos conectados generaron. [THINGSPEAK: 2014]

Sharing Robots

El proyecto Sharing Robots, realizado por el universitario Luis Gutierrez de la Universidad Pública Del El Alto permite (haciendo uso de Scrapping) el control de robots conectados a Internet. Como única funcionalidad, el usuario puede enviar comandos de direcciones a un robot (arriba, abajo, izquierda y derecha) a través de la página web que el mismo proyecto provee. [GUTIERREZ: 2014]

1.2.3. Comparación de productos y proyectos

La Tabla 1 y Tabla 2 muestran una comparativa entre las características de proyectos comerciales y Open-Source relacionados al Internet De Las Cosas.

Producto	**Uso Abierto**	**Uso Exclusivo**	**Comunicación Unidireccional**	**Comunicación Bidireccional**	**Reusabilidad**	**Extensibilidad**	**Panel Web**	**Librerías micro controladores**
Jasper Technologies		X	X	X	X	X	X	?
Xively	X	X	X	X	X	X	X	X
Lockitron		X	X	X		X	X	?
Nest Labs	X		X	X		X	X	?

Tabla 1: Comparación de productos comerciales

Producto	**Uso Abierto**	**Uso Exclusivo**	**Comunicación Unidireccional**	**Comunicación Bidireccional**	**Reusabilidad**	**Extensibilidad**	**Panel Web**	**Librerías micro controladores**
Meshblu	X	X	X	X	X	?		X
ThingSpeak	X	X	X		X	X	X	
Sharing Robots	X	X	X				X	

Tabla 2: Comparación de proyectos Open-Source

1.3. Planteamiento del Problema

Los resultados que se obtienen en la investigación sobre proyectos Open-Source relacionados al Internet De Las Cosas (ver Anexo A), muestran que entre los proyectos existentes, una gran mayoría quedan como Prueba de Concepto[7], es decir su desarrollo ya no continúa, además de no contar con características que las plataformas comerciales existentes brindan (ver sección 1.2.1).

1.3.1. Problema General

Entre los proyectos Open-Source de tipo plataforma relacionados al Internet De Las Cosas, un subconjunto de estos no cuenta con varias funcionalidades (gestión de usuarios, gestión de permisos sobre dispositivos, librerías para micro controladores y otros) a diferencia de las plataformas comerciales existentes.

1.3.2. Problemas Secundarios

a) La cantidad de proyectos Open-Source relacionados al Internet De Las Cosas que mantengan un desarrollo continuo es baja comparada a otro tipo de proyectos. (ver Anexo A)

b) Algunos proyectos Open-Source de tipo plataforma relacionados al Internet De Las Cosas solo ofrecen los canales de comunicación como característica principal.

c) Algunos proyectos Open-Source de tipo plataforma relacionados al Internet De Las Cosas no brindan al desarrollador librerías para micro controladores que faciliten la

7 Prueba de Concepto o Proof Of Concept (idioma inglés), hace referencia a una evidencia que demuestra que una idea de negocio o cualquier otro tipo de idea es factible.

comunicación con la plataforma que plantean.

d) Algunos proyectos Open-Source relacionados al Internet De Las Cosas quedan como "Prueba de Concepto", es por ello que su desarrollo ya no continúa. (Ver Anexo A)

e) Varios experimentos de hardware que requieren conectividad con Internet implementan plataformas de prueba que carecen de reusabilidad y escalabilidad. [HELCEN: 2012]

1.4. Objetivos

1.4.1. Objetivo Principal

Diseñar y desarrollar una plataforma Open-Source para el Internet De Las Cosas, que además de proveer canales de comunicación, cuente con las funcionalidades que una plataforma comercial ofrece (gestión de usuarios, gestión de permisos sobre los dispositivos y librerías para micro controladores).

1.4.2. Objetivos Específicos

- Diseñar una arquitectura escalable con la capacidad de ser extendida y reutilizable por otros desarrolladores con facilidad.
- Visualizar los datos que generen los dispositivos a través de la plataforma web.
- Brindar un mecanismo que permita al usuario personalizar los canales de comunicación a medida que este lo requiera.
- Crear librerías para micro controladores que faciliten la comunicación con la plataforma a través de los canales de comunicación.

1.5. Justificación

1.5.1. Justificación Técnica

En los últimos años, diferentes tecnologías de desarrollo surgen, pensar en comunicación en tiempo real a través de la web ya es factible por las tecnologías existentes (frameworks, librerías, protocolos y otros). [LEGGETTER: 2013]

Por otro lado, la conexión de dispositivos de hardware a Internet es totalmente factible por la existencia de una gran variedad de módulos de comunicación (WiFi, Ethernet, Bluetooth y otros) como también la existencia de sistemas operativos empotrados (Raspberry Pi, BeagleBone y otros).

1.5.2. Justificación Social

El presente trabajo al ser una alternativa Open-Source, cuenta con la posibilidad ser adaptado a las necesidades de una empresa, de un usuario o de un grupo de personas en específico. El presente proyecto tiene la libertad de ser distribuido, adaptado, utilizado y mejorado, además de estar publicado bajo una licencia Open-Source.

1.5.3. Justificación Económica

Los proyectos Open-Source trabajan en equipos, con la diferencia que los miembros por lo general, se encuentran alrededor del mundo. Tanto desarrolladores, traductores y usuarios generan una retro alimentación encontrando errores, traduciendo a otros idiomas, realizando nuevas características y estas se encuentran disponibles públicamente.

El presente proyecto al ser un software que debe ser instanciado en un servidor, el costo económico dependerá mucho de la cantidad de dispositivos que se deseen conectar. A más usuarios y dispositivos se conecten, la infraestructura de servidores también se incrementará.

1.6. Alcances y Límites

1.6.1. Límites

Los límites del presente proyecto son:

- El prototipo es probado solamente en servidores GNU/Linux específicamente en distribuciones basadas en Debian (Ubuntu, Mint y otros).
- Se crean librerías para micro controladores ATMEGA-328 y dispositivo Raspberry Pi solamente.
- La plataforma solo puede ser utilizada por dispositivos y sistemas los cuales tengan

conexión a Internet o caso contrario a la red donde la plataforma se encuentre instanciada.

- El prototipo trabaja con tipos de dato básicos, como ser cadenas de caracteres, números enteros, números reales, geolocalización e imágenes codificadas en base64.
- La plataforma no cuenta con el control de estado del dispositivo, es decir, no dará un diagnóstico si el dispositivo se encuentra o no en línea.

1.6.2. Alcances

La plataforma brinda gestión de usuarios, permisos sobre dispositivos, canales de comunicación, mecanismos de autorización a estos canales y librerías para micro controladores y sistemas embebidos. Tanto desarrolladores como usuarios pueden interactuar con la plataforma a través de una Interfaz de Programación de Aplicaciones RESTful (RESTful API) y una Plataforma Web respectivamente. Los canales de comunicación soportan protocolos MQTT y WebSocket para la comunicación con dispositivos y usuarios en tiempo real. La Figura 1 muestra la arquitectura de toda la plataforma.

1.7. Metodología

Para la fase de investigación se hace uso de la Metodología Sistémica ya que este enfoque brinda a la Informática una herramienta conceptual y de acción lo cual permite pasar de lo teórico a lo práctico. Las principales etapas de esta metodología son:

a) Reconocimiento de problemas sistémicos: extracción de los aspectos relacionales y estructurales del problema.

b) Abstracción del problema: identificación del problema dentro de un marco conceptual particular.

c) Aplicación propia: uso de una herramienta metodológica apropiada para resolver el problema en su formulación abstracta.

d) Interpretación de los resultados en términos del problema específico.

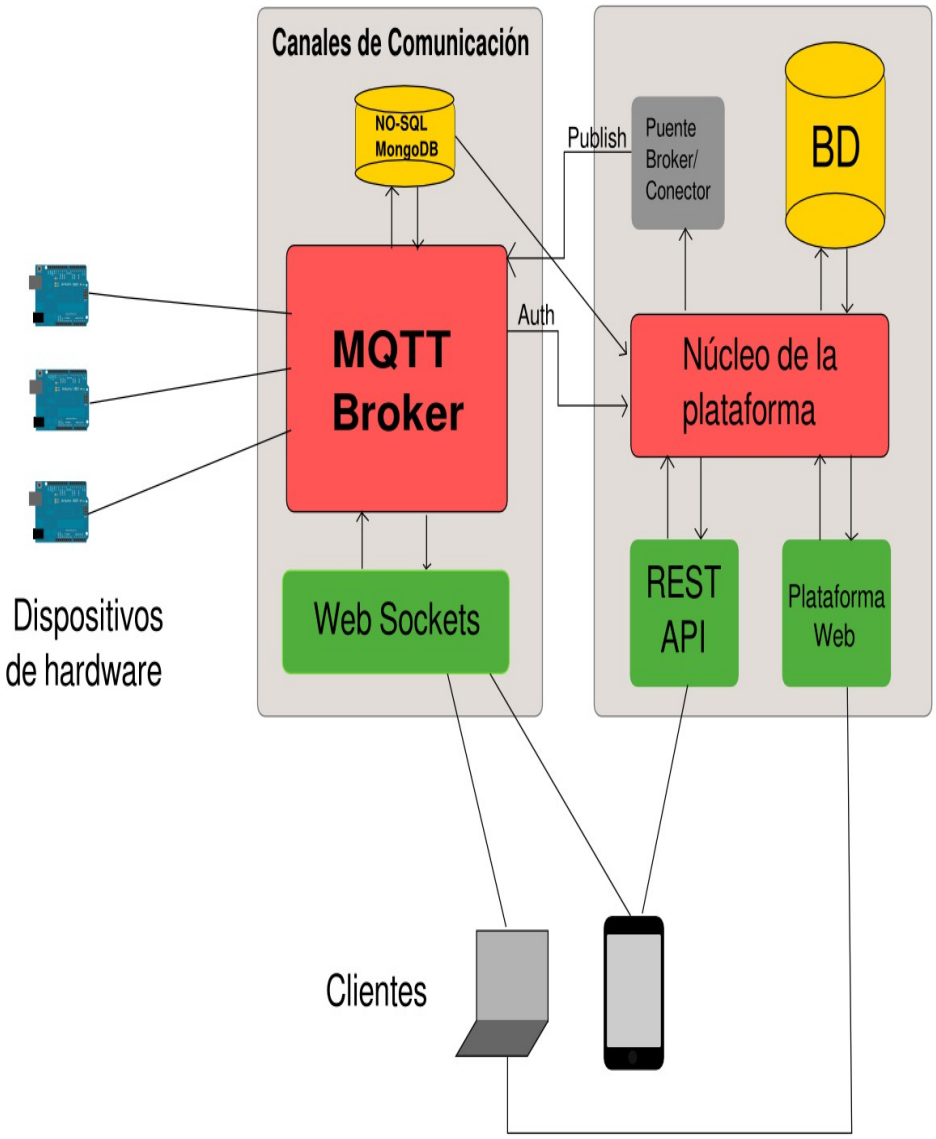

Figura 1: Arquitectura plataforma Beehive

CAPÍTULO 2
MARCO TEÓRICO

2.1. Internet De Las Cosas

Según [HÖLLER: 2014] el Internet De Las cosas puede ser definido de la siguiente forma: "*El Internet De Las Cosas pretende ofrecer una conectividad avanzada entre dispositivos, sistemas y servicios que va más allá de una comunicación M2M (Machine to Machine) y que cubre una variedad de protocolos, dominios y aplicaciones*".

Teniendo en cuenta el concepto anterior, es posible definir la siguiente formula par el Internet De Las Cosas. [MCEWEN, CASSIMALLY: 2014]

Objeto Físico + Controllador, Sensor y Actuador + Internet = Internet De Las Cosas

Hoy en día por la gran facilidad de acceso a Internet que se tiene en una gran parte del mundo ya es posible tener la idea de poder conectar otros dispositivos más allá de teléfonos móviles, tablets, computadoras de escritorio y otros. Actualmente el decir que televisores, termostatos, refrigeradores y otros se encuentran conectados a través de Internet ya es un hecho. La tecnología avanzó lo suficiente tanto a nivel de hardware y software para poder conectar con facilidad cualquier dispositivo electrónico a Internet.

2.1.1. El impacto del Internet De Las Cosas

Con el pasar del tiempo el interés hacia el Internet De Las Cosas crece, la empresa Gartner realiza un estudio en el que se estima que para el año 2020 la cantidad de dispositivos electrónicos conectados (excluyendo computadoras personales, teléfonos inteligentes, tablets) al Internet De Las Cosas será aproximadamente de 26 billones. Además que los proveedores de productos y servicios para el Internet De Las Cosas generará un ingreso de aproximadamente 300 billones de dólares. [GARTNER: 2013]

Al ser el Internet De Las Cosas un concepto grande, se estima que gran parte de los ingresos que generará el Internet De Las Cosas será por parte de los servicios que se brinden y no por

los productos de hardware creados. Al ser los productos de hardware los que generarán demasiada información (sensores) se necesitarán varios desarrolladores para poder procesar dicha información. [ASAY: 2014]

El estudio realizado por Vision Mobile muestra una distribución global de los desarrolladores para el Internet De Las Cosas (ver Figura 2).[SCHUERMANS: 2014]

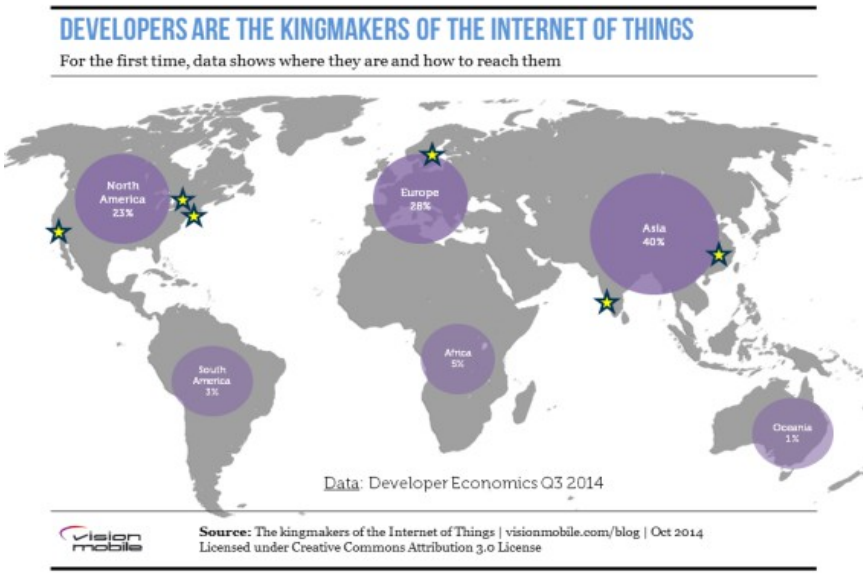

Figura 2: Distribución desarrolladores para Internet De Las Cosas

La tabla muestra en texto los valores identificados en la Figura 2.

Región	**Porcentaje de Desarrolladores**
América del Norte	23%
América del Sur	3%
Europa	28%
Asia	40%
África	5%
Oceanía	1%

Tabla 3: Distribución de desarrolladores para el Internet de las Cosas

Estos indicadores muestran que no existe un área que domine totalmente el campo del Internet De Las Cosas en términos de desarrolladores lo cual sirve como indicador para mostrar que el

nicho para emprendimientos relacionados al Internet De Las Cosas puede encontrarse en cualquier región donde existan desarrolladores para el Internet De Las Cosas.

2.1.2. Aplicaciones para el Internet De Las Cosas

Al existir diferentes tipos de dispositivos que podrán interactuar con seres humanos se generará una cantidad increíble de datos (BigData) y el uso apropiado de estos datos puede ser aplicado para el beneficio del ser humano. Entre algunas áreas que serán beneficiadas por el Internet De Las Cosas son la agricultura, salud y domótica. [DACOSTA:2013]

Agricultura

El manejo y control de la industria agrícola es dificultoso ya que existen muchos factores en juego (factores naturales y factores humanos). Hoy en día a pesar que existe un gran interés de inversionistas sobre la agricultura y la tecnología es un hecho no muy conocido que la agricultura fue y sigue siendo uno de los sectores que proveerá al Internet De Las Cosas una adopción a gran escala. [MOHAMMED: 2014]

La siguiente lista muestra algunas áreas de la agricultura que serán campos de prueba para el Internet De Las Cosas:

a) *Productividad* – El campo de agricultura de precisión (una actividad que que usa el análisis de datos para optimizar decisiones agrícolas) es un gran campo de oportunidades para el Internet De Las Cosas. En la actualidad, es más crítico que nunca el maximizar el rendimiento de cada acre de tierra que se encuentra dedicado a la producción de alimentos. Con la ayuda de dispositivos inalámbricos conectados a la nube será posible maximizar la producción de la cosecha automatizando operaciones de agricultura (riego, control de humedad y otros) además de proveer un monitoreo en tiempo real y el análisis de datos para una toma de decisiones inteligente.

b) *Control de Plagas* – A medida que el movimiento orgánico gana más popularidad, el interés por alternativas más efectivas y relativamente económicas a los pesticidas por parte de la industria agrícola y de alimentos ha aumentado considerablemente. Una de

estas alternativas son las trampas de feromonas que, combinadas con el Internet De Las Cosas pueden resultar útiles ya que será posible monitorear la cantidad de alguna plaga y en caso que esta población aumente, se podrá activar una trampa de feromonas para poder acabar con dicha plaga.

c) Conservación del Agua – Desde el punto de vista de la escasez del agua, la agricultura históricamente ha sido un reto que demanda un gran conocimiento técnico, dominio en colección de datos y de sistemas de riego. Para una respuesta efectiva, granjeros requieren información precisa en tiempo real que ayude a minimizar el desperdicio, exceso y falta de irrigación, de esta forma es posible manejar los costos del agua de una forma más efectiva. Combinando el Internet De Las Cosas con dispositivos inalámbricos y sistemas de monitoreo del suelo, granjeros pueden medir humedad, detectar fugas de agua y controlar con más eficiencia el uso de energía, todo en tiempo real.

Salud

En países de Europa o Norte América, existen nuevos dispositivos que permiten realizar el diagnóstico y tratamiento. Algunos de estos dispositivos se encuentran conectados a Internet y al ser así, se convierten en un importante elemento para el Internet De Las Cosas. Entre algunos beneficios de tener conectado equipamiento de prueba de laboratorio y otros dispositivos de diagnóstico se tiene: [ISAACS: 2014]

a) *Reducir el Tiempo de inactividad a través de monitoreo y soporte remoto* – Un dispositivo conectado a Internet puede ser probado y diagnosticado remotamente, como ejemplo un técnico puede conectarse desde su oficina y realizar el diagnóstico de un equipo de imagen por resonancia magnética que este fallando. El técnico podría encontrar la raíz del problema además que este técnico puede conectarse con los técnicos del hospital al que pertenece el equipo para realizar el soporte técnico. Cuando la causa del problema sea identificada, una nueva pieza puede ser entregada incluyendo las instrucciones para reemplazarla.

b) *Cumplimiento proactivo mediante la reposición de suministros antes de que sean nece-*

sarios – Un equipo médico conectado a Internet puede reportar cuando componentes críticos de funcionalidad se encuentran en niveles bajos. Un ejemplo son los niveles de helio en un equipo de imagen por resonancia magnética que necesitan ser monitoreados para asegurar que el equipo se encuentra funcionando correctamente. Esto permite que técnicos en el área reciban notificaciones de visitar un hospital antes que los niveles de helio de estos dispositivos se hayan agotado, evitando así, el apagado total del equipo además de re-programar exámenes a los pacientes.

c) *Programación eficiente para atender más pacientes* – En el campo médico, existe una gran cantidad de equipos que son muy caros como para ser manejados de manera ineficiente. Un equipo médico conectado a Internet puede proveer estadísticas diarias de uso que pueden ser aprovechadas para la programación de pacientes. Siguiendo con el ejemplo del equipo de imagen por resonancia magnética, en algún área este equipo podría ser utilizado solo en un 20 por ciento mientras que en otra área este mismo equipo puede encontrarse sobre-utilizado. Siendo el caso, doctores pueden reasignar a los pacientes a utilizar el otro equipo. Esto a través de una aplicación basada en la nube que realice la programación de uso de dichos equipos.

Domótica

Por muchos años el mercado de domótica fue dominado por empresas que realizan instalaciones en el hogar, equipando clientes adinerados con instalaciones entre $10.000 y $100.000 para poder controlar luces, seguridad, aire acondicionado, el audio de la casa y otros. En la actualidad surgen nuevas formas de automatización más simples gracias a los productos DIY (Do it yourself – Hazlo por ti mismo) que existen en el mercado haciendo innecesarios varios servicios de las empresas de automatización del hogar. Algunas marcas que proveen esto son Nest, Yale, Iris, Apigy, Sonos, Lutron y Belkin que individualmente controlan la corriente A/C, luces, cerraduras, puertas de garaje, cortinas y cualquier cosa que pueda conectarse. [MOOREHEAD: 2013]

Al crecer un mercado más amplio de productos DIY un nuevo reto surge. Un usuario puede

comprar una variedad de estos productos (termostatos Nest, cerraduras de Apigy, conectores Belkin, interruptores de luz Lutron) entonces este tendrá la obligación de utilizar cuatro aplicaciones diferentes (una por producto DIY) además de adaptarse al funcionamiento de las plataformas que pueden diferir unas de otras. En adición, no existen formas fáciles para que estos dispositivos trabajen como un solo sistema, es decir, programación de eventos y otros ya que estos productos requieren aplicaciones distintas. Una de las soluciones a este problema es la de crear una aplicación superior para las aplicaciones que difieren. [MOOREHEAD: 2013]

2.1.3. Internet De Las Cosas y Open-Source

El software Open-Source y los estándares abiertos ya se encuentran presentes en el contexto del Internet De Las Cosas. Una cantidad considerable de aplicaciones que obtienen y analizan datos dependen de software Open-Source y estándares abiertos. Los dispositivos de hardware con software embebido en el contexto del Internet De Las Cosas ejecutan un sistema operativo GNU/Linux en su gran mayoría, es por ello que los fabricantes de hardware adoptaron de un 40 a 50 por ciento sistemas operativos Open-Source, sin embargo, existe un gran porcentaje de software propietario dentro de estos sistemas embebidos. [KOROLOV: 2015]

La gran diferencia entre software Open-Source y software propietario no radica en la tecnología que se encuentre dentro de los dispositivos conectados ni en las aplicaciones que los controlen, la verdadera diferencia radica en un punto intermedio, en como los dispositivos mantendrán comunicación, los estándares de mensajería y las aplicaciones que implementen estos estándares.

Existen diferentes alternativas tanto propietarias como abiertas para estos estándares de comunicación. Por un lado se encuentra la Fundación Eclipse que pretende crear estándares abiertos para el Internet De Las Cosas además de plataformas Open-Source (ver sección 1.2). Por otro lado, existen diferentes nichos de productos propietarios tales como la plataforma Thread de Google y el Kit del Hogar de Apple.

La adopción de estándares abiertos y software Open-Source, ayudará a los ecosistemas del Internet De Las Cosas a crecer y desarrollarse haciendo que los productos creados tengan facili-

dad de comunicación entre ellos.

2.2. Dispositivos de Hardware

2.2.1. Módulos de conexión a Internet

En la actualidad el mercado brinda diferentes módulos de hardware que permiten a un micro controlador efectuar una conexión a Internet. Estos módulos permiten conectarse a través de diferentes tecnologías de conexión (Ethernet, Wifi, Red GSM y otros)

A continuación se describen algunos módulos de hardware populares en el mercado que permiten la conexión a Internet.

Módulo Ethernet para Arduino

El módulo Ethernet (Shield Ethernet) permite a un dispositivo Arduino conectarse a una red local o a Internet (ver Figura 3). Basado en el chip ethernet Wiznet W5100, este módulo permite trabajar con protocolos TCP y UDP soportando a lo mucho cuatro conexiones simultáneas. Este módulo posee un zócalo para tarjetas micro-SD, el cual puede ser utilizado para almacenar archivos para compartir en la red. Los datos de esta tarjeta micro-SD pueden ser utilizada por la librería SD que provee Arduino. [ETHERNETSHIELD: 2013]

La empresa fabricante de Arduino brinda las librería para poder trabajar con el módulo ethernet y la tarjeta micro-SD como software Open-Source.

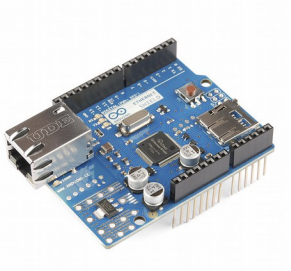

Figura 3: Módulo Ethernet para Arduino

Módulo WiFi CC3000

Creado por la empresa Texas Instruments, el CC3000 es un procesador de red inalámbrico que simplifica la conexión a Internet a través de una red inalámbrica (ver Figura 4). Esta solución WiFi minimiza los requerimientos de software del micro controlador anfitrión y por lo tanto es una solución ideal para aplicaciones embebidas que corren bajo micro controladores de precio y consumo bajo.

Una de las características que resaltan de este módulo es la de configuración inteligente, la cual permite configurar los datos de la red inalámbrica a la cual se desee conectar a través de un smart phone, tablet o PC, evitando así la necesidad de tener información de una red WiFi pre establecida en micro controlador anfitrión. [CC3000: 2012]

Figura 4: Módulo CC3000 de Texas Instruments

Módulo SIM5218 para redes 3G

El módulo SIM5218 (ver Figura 5) permite conectarse a redes celulares de alta velocidad WCDMA y HSPA para hacer posible la creación del siguiente nivel de interactividad de proyectos. Junto con la conexión a redes celulares, este módulo incluye un GPS[8] que permite obtener la ubicación del dispositivo en ambientes abiertos y cerrados. Además de ser utilizado como un módulo por un micro controlador, este permite ser utilizado como un modem 3G en una PC. Entre otras características, el SIM5218 permite al fabricante hacer uso de tarjetas de almacenamiento SD, cámara y un kit de audio de tal forma que este módulo pueda ser convertido con facilidad en un teléfono celular. [SIM5218: 2013]

La comunicación con el hardware se realiza por comandos AT los cuales son enviados al mó-

8 GPS – Global Positioning System

dulo por puerto serial. Estos comandos son cadenas de texto finalizadas por un carácter de retorno de carro que indican al módulo que operación realizar, tal como, una llamada, obtener datos del GPS, conectarse a Internet, crear una conexión TCP o UDP, envío de SMSs y otros. [SIM5218-AT: 2011]

Figura 5: Módulo 3G SIM5218

2.2.2. Micro controlador ATMega328 y Arduino UNO

Creado por la empresa ATMEL el modelo ATMEGA328 es un micro controlador que pertenece a la familia de micro controladores megaAVR. El ATMega328 cuenta con una arquitectura RISC[9] de 138 instrucciones (la mayoría de ellas dura un ciclo de reloj) y con un voltaje de trabajo de 1.8 a 5.5 voltios. [ATMEGA: 2014]

La Tabla 4 muestra las principales características del micro controlador ATMega328

Característica	**Valor**
Memoria Flash ISP	32Kb
EEPROM	1K
SRAM	2Kb
Cantidad de Pines	28
Máxima frecuencia de operación	20 MHz
CPU	8 bits
Máximo de pines de entrada y salida	26

Tabla 4: Características micro controlador ATMEGA-328

El modelo ATMega328 es utilizado en varios proyectos en los que el bajo consumo y costo

9 RISC – Reduced Instruction Set Computing, una arquitectura para microprocesadores

son necesarios. Una de las implementaciones más comunes de este micro controlador es la plataforma de desarrollo de hardware Arduino.

Arduino

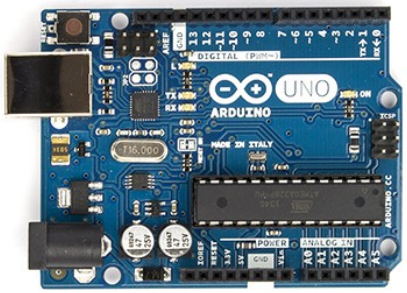

Figura 6: Placa Arduino UNO

Arduino es un plataforma física Open-Source basada en una placa electrónica simple, con la lógica de "Hardware y Software fácil de utilizar" además de un entorno de desarrollo para crear software para esta placa. Esta puede ser utilizada para crear diferentes tipos de proyectos que incluyan lectura de diferentes tipos de sensores o el control de diferente tipos de salidas físicas (luces, motores y otros). Al ser hardware Open-Source, el esquema de Arduino puede ser utilizado para crear clones de esta placa. Actualmente existe una gran variedad de clones los cuales son construidos bajo los esquemas de un Arduino, al ser así, estos clones pueden ser utilizados como si fuera un Arduino Original. [ARDUINO: 2014]

Arduino cuenta con diferentes modelos (Arduino Uno, Arduino Leonardo, Arduino Mega, Arduino Nano y otros).

Arduino Uno (ver Figura 6) es considerado el modelo más popular dentro de la familia de Arduino, esta placa usa el micro controlador ATMega328, cuenta con 14 pines de entrada y salida digital, 6 entradas analógicas, un resonador cerámico de 16 Mhz, conexión USB, un alimentador de poder, botón de reset y cabecera ICSP.

Este modelo se diferencia de las demás placas al no hacer uso del chip FTDI USB que realiza

la comunicación por protocolo serial. En vez de este chip usa un micro controlador ATMega8U2 para realizar conversión USB-Serial. [ARDUINO-UNO: 2014]

2.2.3. Raspberry Pi 1

Este dispositivo es una computadora de bajo costo además de tener el tamaño de una tarjeta de crédito que fue creada por la Fundación Raspberry Pi en Inglaterra. Esta computadora puede conectarse con monitores de computadora e incluso televisores, además de utilizar teclado y ratón estándar. Este pequeño dispositivo fue creado con el propósito de que todo tipo de personas exploren el uso de la computadora, además de aprender a programar una computadora en lenguajes como Scratch y Python. Entre sus características, esta computadora puede puede realizar las mismas tareas que se realizarían en una computadora de escritorio cualquiera (multimedia, ofimática, juegos y otros). [RASPBERRYPI: 2012]

El Raspberry Pi 1 fue lanzado al público el año 2012, cuenta con tres modelos (modelo A, modelo B y modelo B+) los cuales hacen mejoras a modelos anteriores como ser más puertos USB, mayor cantidad de pines de programación, mayor memoria RAM (modelo B y B+).

La Tabla 5 muestra las características de hardware del último modelo B+ para el Raspberry Pi 1 [RASPBERRYPIBPLUS: 2014]

Característica	**Valor**
Precio	Modelo A: 25 USD Modelo B y B+: 35 USD
CPU	Procesador de un solo núcleo ARM1176JZF-S 700MHz
RAM	Modelo A: 256MB Modelo B y B+: 512MB
Almacenamiento	Slot MicroSD
Gráficos	Broadcom Video Core IV
Energía	Modelo A: 1.5W Modelo B: 3.5W Modelo B+: 3.0W
GPIO	40 pines I/O

Puertos USB	4 puertos
Red	Conector Ethernet

Tabla 5: Características de hardware Raspberry Pi B+

El Raspberry Pi 1 modelo B+ (ver Figura 7) al tener características de hardware inferiores a una computadora de escritorio normal, no es posible instalar el sistema operativo Windows ya que los requerimientos de las versiones soportadas actualmente minimamente requieren 1GB de memoria RAM, es por ello que la mayoría de sistemas operativos para esta computadora se encuentran basados en distribuciones basadas en Unix (GNU/Linux, FreeBSD, OpenBSD).

La Tabla 6 muestra una lista de las distribuciones GNU/Linux más populares para el Raspberry Pi 1.

Nombre de Distribución GNU/Linux	**Descripción**
Raspbian	Un clon de la distribución Debian diseñada para el Raspberry Pi 1. Cuenta con versión de escritorio y versión de servidor
ArchLinux ARM	Una compilación de la distribución ArchLinux para procesadores ARM

Tabla 6: Distribuciones más populares para el Raspberry Pi 1

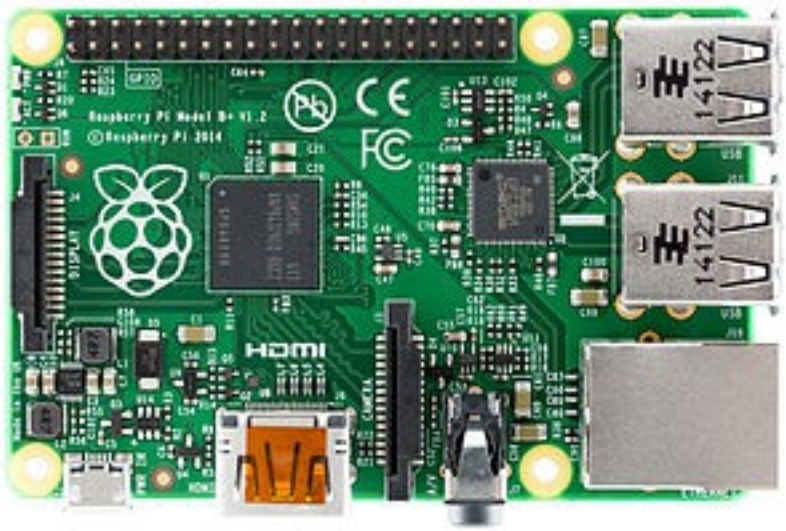

Figura 7: Raspberry Pi 1 modelo B+

2.3. Arquitecturas y Patrones de Software

2.3.1. Arquitectura Orientada a Recursos

ROA (Resource Oriented Architecture) es un estilo de arquitectura de software para diseñar y desarrollar un producto de tal forma que la información publicada se encuentre en forma de recursos y sea accesible a través de interfaces RESTful (ver sección 2.4.1.1.). Esta arquitectura toma como base el concepto de recurso, el cual es un componente que puede ser accedido directamente, además que este puede ser manejado a través de un protocolo común (Protocolo HTTP). [RICHARDSON, RUBY: 2007]

Las plataformas RESTful basadas en REST permiten la creación de una Arquitectura Orientada a Recursos.

La Tabla 7 muestra los conceptos principales de esta arquitectura.

Término	Definición
Recurso	Cualquier entidad dentro del sistema que tenga la importancia suficiente para ser publicado como tal
Nombre de Recurso	Identificación única del recurso
Representación del Recurso	Información útil que indica el estado actual del recurso
Enlace al Recurso	Enlazar a otra representación del mismo recurso o de otro recurso
Interfaz de Recurso	Interfaz uniforme para acceder y manipular el estado de un recurso

Tabla 7: Conceptos de la arquitectura orientada a recursos

Al momento de diseñar un software que deba ser realizado bajo una arquitectura orientada a recursos es importante considerar las siguientes etapas:

a) Considerar que será y que no será un recurso. Una vez definidos los recursos estos deben ser accedidos a través a una interfaz común.

b) Definir el contenido del mensaje al invocar los métodos como también las respuestas

de un recurso.

c) Definir un esquema de direcciones (URI[10]) para poder acceder a instancias de recursos.

d) Definir el esquema de respuesta de los recursos.

e) Relacionar las funciones del recurso con los métodos del protocolo HTTP (ver sección 2.3.2.)

2.3.2. Servicios RESTful

En la actualidad varios protocolos denominados como Servicio Web (SOAP, WPS y otros) realmente no se encuentran relacionados con la Web como tal. Opuestamente a la simplicidad que propone la Web, estos Servicios Web proponen arquitecturas pesadas para el acceso distribuido a objetos. De este modo, este tipo de arquitecturas para Servicios Web, inventan nuevamente e incluso ignoran muchas de las características que hicieron a la Web exitosa.

Partiendo del punto en el que se describe a que todo es un servicio (sitio web, aplicación web) es posible formar la siguiente analogía: Las características que hacen a un sitio web fácil de utilizar para un usuario también hacen que una API expuesta como servicio web sea fácil de utilizar para un programador.

REST (Representational State Transfer) es una abstracción de la arquitectura de la World Wide Web. Este es un estilo de arquitectura que contempla todos los componentes dentro de un sistema hiper media distribuido. [FIELDING: 2000]

La Web funciona gracias al protocolo HTTP[11], ya que este brinda muchas facilidades al cliente que realiza una conexión con un servidor web. El protocolo HTTP permite realizar diferentes acciones en los diferentes recursos gracias al uso de métodos, de este modo el servidor interpreta una petición en base al método que esta presente. La Tabla 8 muestra los métodos del protocolo HTTP más utilizados. [HTTP: 1999]

10 URI - Universal Resource Identifier, una composición de URL + URN (uniform resource name)
11 HTTP – Hypertext Transfer Protocol

Método	Descripción
GET	Indica que la petición solo desea obtener cierta información, esta puede ser en base a parámetros que son incluidos en la URI[12]
POST	Indica que la petición desea persistir la información enviada a través de una operación de inserción
PUT	Indica que la petición realizará cambios a un recurso existente. Se pueden entender como una petición que actualizará un recurso.
DELETE	Indica que la petición eliminará un recurso existente

Tabla 8: Métodos del protocolo HTTP

Por lo tanto un servicio RESTful es una instancia que hace uso de los conceptos pertenecientes a una arquitectura orientada a recursos, es decir, hace uso del protocolo HTTP para las diferentes acciones que se realicen con un recurso.

La Tabla 9 muestra un ejemplo de un servicio RESTful en el que publica un recurso "personas" y define un esquema de respuesta para los diferentes métodos del protocolo HTTP.

Método / URI	Descripción	Respuesta
GET http://example.com/personas/	Obtiene una lista de personas (recursos)	[{"nombre":"Juan Perez"}, {"nombre":"Maria Sanchez"},]
GET http://example.com/personas/Juan_Perez	Obtiene una persona específica	{ "nombre":"Juan Perez" }
POST http://example.com/personas	Adiciona una nueva persona en el servidor	{ "status": "ok" }
PUT http://example.com/personas/Juan_Perez	Modifica un recurso existente en el servidor	{ "nombre":"Juan Perez Olguin" }
DELETE http://example.com/personas/Juan_Perez	Elimina un recurso existente en el servidor	{ "status": "eliminado" }

Tabla 9: Ejemplo servicio RESTful

12 URI - Universal Resource Identifier, una composición de URL + URN (uniform resource name)

2.3.3. Patrón Publish/Subscribe

El envío de mensajes haciendo uso del patrón Publish/Subscribe permite que varios usuarios puedan enviar mensajes a una entidad en el servidor, esta entidad por lo general se llama tópico y cada tópico puede tener varios suscriptores. Cada suscripción recibe una copia de cada mensaje que se envía al tópico suscrito. [HIVEMQ: 2015]

Este patrón es una alternativa a un modelo cliente-servidor tradicional en la que el cliente se comunica directamente con un punto final (endpoint). El objetivo principal de desacoplar a un cliente el cual envía un mensaje al servidor (publisher) con uno o más clientes (subscribers) que reciben mensajes a través de un tópico. Esto significa que el usuario que envía el mensaje (publisher) y los usuarios que reciben este mensaje (subscribers) no tienen un conocimiento directo de la existencia del usuario remitente (publisher). Para que toda esta lógica de transferencia de mensajes se realice, es necesario un "broker". Tanto remitentes y suscriptores conocen a este "broker" el cual se encarga de filtrar todos los mensajes en base al tópico y hace el envió a los usuarios que se encuentren suscritos a este tópico.

Desacomplamiento

Para realizar el desacoplamiento entre remitentes y suscriptores el broker contempla tres dimensiones:

a) Espacio – Remitentes y Suscriptores no necesitan conocerse entre sí (dirección IP o puerto).

b) Tiempo – Remitentes y Suscriptores no necesitan estar en ejecución o conectados al mismo tiempo.

c) Sincronización – Las operaciones en ambos elementos (remitentes y suscriptores) no son interrumpidas durante la publicación o suscripción.

Gracias a este desacoplamiento entre remitentes y suscriptores es posible mejorar la escalabilidad. Esta escalabilidad se debe a que las operaciones en el "broker" pueden realizarse en paralelo y ser procesadas en un entorno basado en eventos. Agrupando varios nodos del "broker"

para distribuir la carga en servidores individuales con balanceadores de carga permiten que el patrón publish/subscribe pueda mantener miles de conexiones concurrentes.

Filtrado de Mensajes

Para que el "broker" pueda filtrar los mensajes que lleguen a este, es posible (dependiendo del caso) hacer uso de las siguientes opciones de filtrado.

a) Basados por un tópico – Los mensajes se filtran en base al tópico que presenten, entonces los suscriptores reciben los mensajes a través del broker en base a los tópicos a los que se hayan suscrito. Los tópicos por lo general son cadenas de caracteres que en algunos casos tienen una estructura de jerarquía.

b) Basados por el contenido – El broker realiza el filtrado en base al contenido del mensaje, al ser así, esto evita que el contenido pueda ser encriptado o cambiado con facilidad.

c) Basados por el tipo – Generalmente en lenguajes orientados a objetos es una práctica común filtrar los mensajes en base al tipo o clase. Siendo así, un suscriptor puede esperar que el broker filtre todos los mensajes de cierta clase (clase Exception por ejemplo).

Hacer uso del patrón publish/subscribe brinda ventajas como la escalabilidad gracias al desacoplamiento entre remitentes y suscriptores. Esta ventaja genera algunos retos que deben ser tomados en cuenta. En el caso de filtrado por tópico, es importante que tanto remitentes y suscriptores conozcan con anterioridad los tópicos con los que trabajaran. Otro aspecto muy importante es la entrega de mensajes y que el remitente no puede asumir que el mensaje haya sido recibido por algún suscriptor, es posible que el mensaje nunca haya sido recibido por un suscriptor.

2.4. Protocolos de Comunicación

2.4.1. Message Queue Telemetry Transport (MQTT)

MQTT es un protocolo de conectividad máquina a máquina M2M[13] que actualmente es considerado el protocolo para el Internet De Las Cosas. Creado por el Dr. Andy Stanford-Clark de IBM y Arlen Nipper de Arcom en 1999, este protocolo es utilizado en diferentes industrias. El año 2013 este protocolo inicia el proceso de estandarización hasta noviembre de 2014 donde pasa a ser un estándar por parte de la organización OASIS. Sin embargo la especificación del protocolo se encontraba publicada bajo una licencia libre de regalías durante varios años.

MQTT funciona bajo el protocolo TCP/IP y esta diseñado para transportar mensajes de la manera más ligera posible haciendo uso del patrón "publish/subscribe" (ver sección 2.3.3.). Este protocolo es útil para conexiones en lugares remotos en los que el ancho de banda de la red es limitado. Utilizado en sensores es posible realizar la conexión a un servidor mediante un link satelital o incluso conexiones dial-up. [MQTT: 2004]

MQTT esta diseñado para aplicaciones que envíen datos de telemetría[14] o desde sondas espaciales de tal forma que use poco ancho de banda y reduzca el consumo de batería. Es por ello que Facebook lanza la aplicación Facebook Messenger haciendo uso del protocolo MQTT para solucionar problemas de latencia. [ZHANG: 2011]

La especificación de este protocolo consta principalmente de tres puntos: Calidad de Servicio (QoS), Formato de Control de Paquetes y Control de Paquetes. [MQTT-OASIS: 2014]

Calidad de Servicio (QoS)

Los paquetes son entregados de acuerdo a la calidad de servicio QoS definida, MQTT brinda tres tipos de QoS

1. QoS 0 – El mensaje se envía de acuerdo a la disponibilidad de la red. No existe respuesta por el suscriptor y no se realiza un re envío por el remitente. El mensaje llega al

13 M2M – Machine To Machine

14 Telemetría - Tecnología que permite la medición remota de magnitudes físicas y el posterior envío de la información hacia el operador del sistema.

suscriptor ya sea una o ninguna vez.

2. QoS 1 – El servicio se asegura que el mensaje llegue al suscriptor al menos una vez. El paquete envía un sub paquete de agradecimiento para corroborar que el mensaje ha sido enviados.

3. QoS 2 – Utilizado en caso que no se desee la perdida ni la duplicidad de mensajes, el consumo general (tamaño de paquete, consumo de ancho de banda) aumenta con esta calidad de servicio.

Formato de Control de Paquetes

El protocolo MQTT funciona haciendo el intercambio de diferentes paquetes de control, estos paquetes se definen de la siguiente forma:

a) Cabecera obligatoria – Se encuentra en todos los paquetes MQTT y consta de dos bytes. Para definir un paquete se usan los bits del primer byte. Desde el bit 0 al bit 3 se definen las banderas del tipo de paquete que se use y desde el bit 4 al 7 se indica el tipo de paquete: 0x0: Reservado, 0x1: CONNECT, 0x2: CONACK, 0x3: PUBLISH, 0x4: PUBACK, 0x5: PUBREC, 0x6: PUBREL, 0x7: PUBCOMP, 0x8: SUBSCRIBE, 0x9: SUBACK, 0xA: UNSUBSCRIBE, 0xB: UNSUBACK, 0xC: PINGREQ, 0xD: PINGRES, 0xE: DISCONNECT, 0xF: Reservado. El segundo byte es utilizado para la codificación del paquete

b) Cabecera Variable – Algunos tipos de paquetes contienen una cabecera variable que se encuentra entre la cabecera obligatoria y el cuerpo del paquete, esta cabecera consta de dos bytes. Su contenido varia dependiendo el tipo de paquete que se encuentre en la cabecera obligatoria.

c) Cuerpo del Paquete – Después de los dos bytes de la cabecera obligatoria y otros dos bytes de la cabecera variable, el resto del paquete consta del cuerpo del paquete donde se encuentran los datos que se transmiten a través del protocolo. Los siguientes tipos de de paquete requieren un cuerpo: CONNECT, SUBSCRIBE, SUBACK, UBSUBS-

CRIBE y en los siguientes el cuerpo es opcional: PUBLISH. En los tipos de paquetes restantes no es necesario un cuerpo.

Control de paquetes

Después que la conexión TCP se haya realizado por un cliente hacia el servidor, el primer paquete enviado desde el cliente al servidor debe ser un paquete de conexión (CONNECT) con la información de conexión (identificador de cliente, usuario, password y otros) si el servidor vuelve a detectar un paquete CONNECT desde el cliente la conexión es finalizada. Una vez enviado el paquete CONNECT, el servidor responde con un paquete CONNACK que verifica que la conexión se realizó con éxito.

Una vez la conexión se haya realizado existen once opciones que el protocolo MQTT provee: PUBLISH (publicar mensaje), PUBACK (publicación realizada), PUBREC (publicación recibida para QoS 2), PUBREL (publicación liberada para QoS 2), PUBCOMP (publicación completa para QoS 2), SUBSCRIBE (suscripción a tópicos), SUBACK (verificación de suscripción), UNSUBSCRIBE (des suscribirse), UNSACK (des suscripción realizada), PINGREQ (Solcitar ping), PINGRES (respuesta a ping), DISCONNECT (desconectar)

2.4.2. Websockets

El protocolo WebSocket (RFC-6455) permite una comunicación en doble vía entre un cliente ejecutando código no confiable en un entorno controlado (navegador web) y un servidor remoto el cual permite la ejecución de dicho código. El modelo de seguridad de este protocolo es el mismo que es utilizado comúnmente en navegadores web. Inicialmente el protocolo consiste en realizar un "handshake[15]" después del envío de un paquete de datos a través del protocolo TCP. El objetivo principal de este protocolo es el de establecer un mecanismo para aplicaciones basadas en navegador (aplicaciones web) que necesiten una comunicación en doble vía con servidores que no se basan en abrir múltiples conexiones HTTP (por ejemplo polling o long polling) [WEBSOCKET: 2011]

Para hacer posible el uso de websockets, este debe estar implementado tanto en el cliente

15 Handshake – En protocolos de comunicación indica que se ha establecido una conexión válida

como en el servidor, es decir, será necesario una aplicación compatible con el protocolo web-sockets como también un servidor que implemente este protocolo.

La Figura 8 muestra como trabaja el protocolo WebSocket, principalmente los pasos que sigue para realizar la conexión.

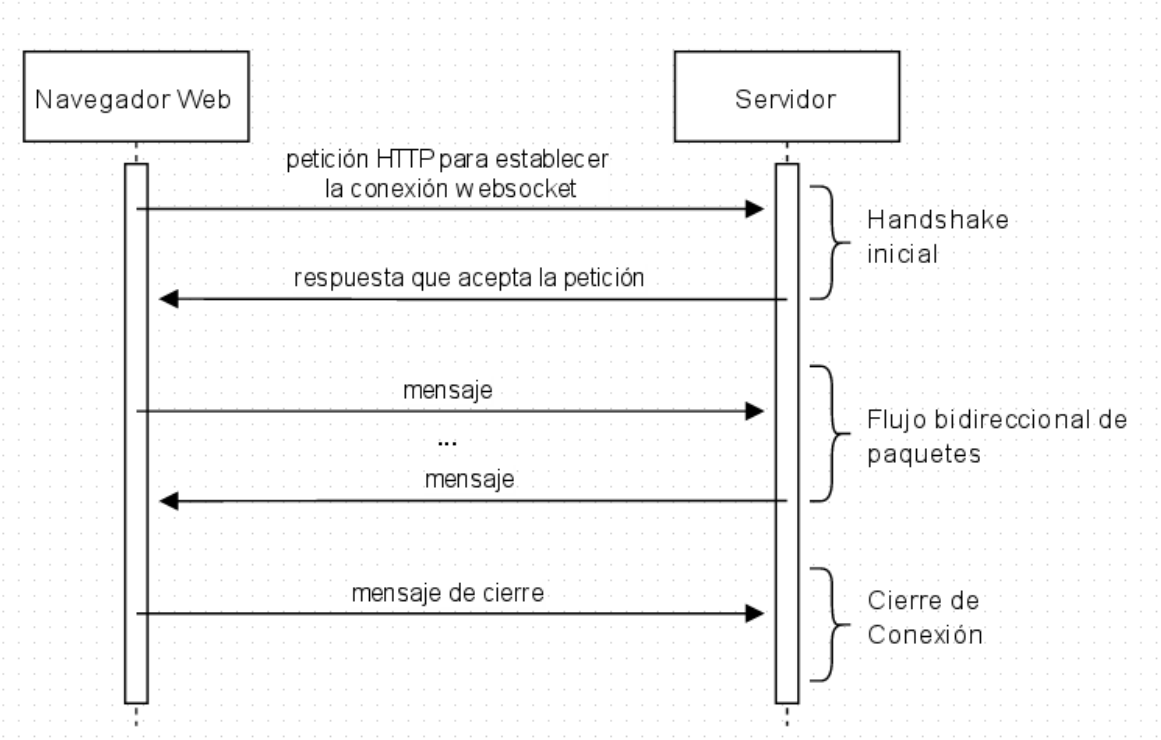

Figura 8: Diagrama de secuencia del protocolo websocket

Hoy en día este protocolo se encuentra implementado en la mayoría de los navegadores en el mercado, lo cual permite a distintos desarrolladores poder implementar el protocolo para los servicios que estos provean. La Tabla 10 muestra la compatibilidad de navegadores y el protocolo WebSocket.

Navegador	**Versión Actual**	**Primera Versión en brindar soporte**
Internet Explorer	11	10
Mozilla Firefox	37	11
Google Chrome	42	16
Safari	8	7

Opera	27	12.1
Safari para iOS	8.3	6.1
Opera Mini	8	No soporta
Android Browser	5	4.4
Opera Mobile	24	12.1

Tabla 10: Compatibilidad de navegadores web y websockets

2.5. Principios de Diseño S.O.L.I.D.

Haciendo uso de metodologías ágiles de desarrollo los cambios en la estructura del proyecto son ineludibles, es por ello que se debe tener una estructura que sea flexible, mantenible y reutilizable.

Algunos síntomas que identifican un diseño pobre son las siguientes:

1. Rigidez – El diseño es difícil de cambiar
2. Fragilidad – El diseño puede colapsar con facilidad
3. Inmovilidad – El diseño es difícil de reutilizar
4. Viscosidad – Es complicado realizar la acción correctamente
5. Complejidad Innecesaria
6. Repetición innecesaria
7. Opacidad. Dificultad al entender el diseño

Los principios de diseño S.O.L.I.D. son el resultado de años de experiencia en ingeniería de software, pero representan la integración de pensamientos y artículos de un gran número de desarrolladores de software e investigadores.

Es recomendable aplicar estos principios de diseño donde se encuentren los síntomas de un mal diseño, sin embargo si un diseño no cuenta con estos problemas, el aplicarlos no sería la mejor opción. [MARTIN&MARTIN: 2007]

2.5.1. Principio “Single Responsability”

Este principio plantea que no debe existir más de una razón para cambiar alguna clase. Es decir, una clase solo debe tener una sola responsabilidad.

El siguiente ejemplo muestra la clase “Rectangulo” que tienes las siguientes funcionalidades: dibujar en una interfaz de usuario y calcular el área. Existen dos aplicaciones que hace uso de esta clase, la primera una aplicación de geometría que necesita obtener el área y la segunda una aplicación gráfica que requiere dibujar la figura.

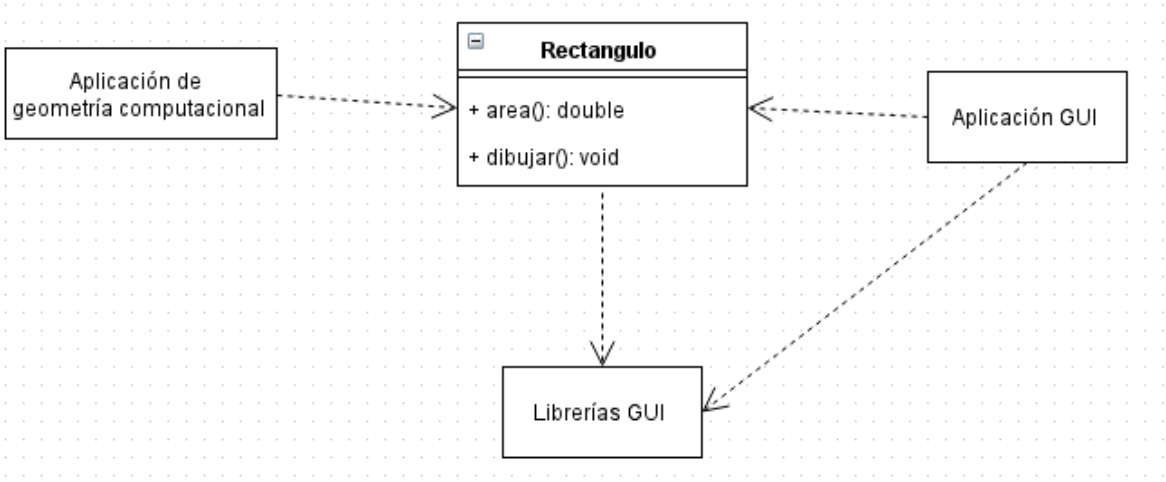

Figura 9: Dos responsabilidades en una clase

La Figura 9 muestra que la clase “Rectangulo” sirve para propósitos matemáticos como también visuales lo cual causa que una aplicación matemática de linea de comandos necesitará incluir sin sentido librerías gráficas. Una posible solución a este problema es el diseño que se muestra en la Figura 10.

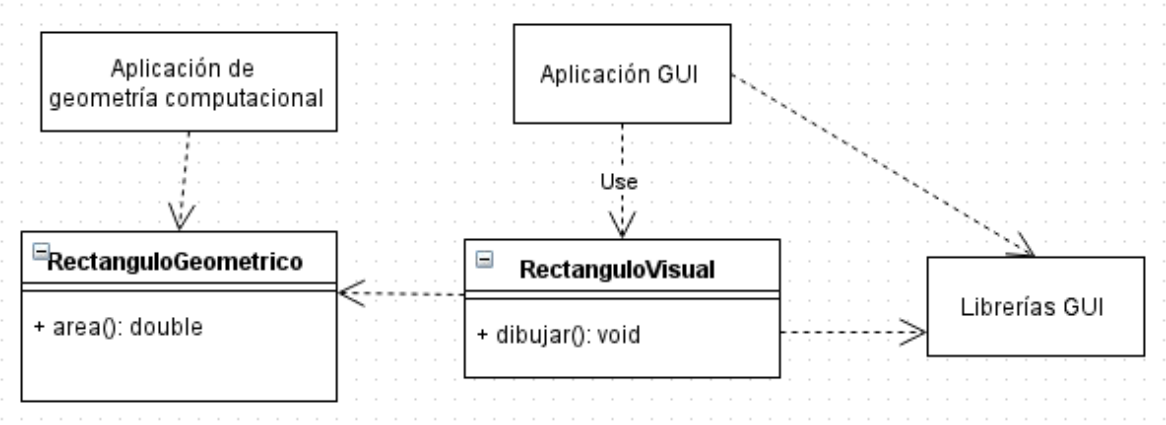

Figura 10: Separación de responsabilidades

De este modo la aplicación en línea de comandos no requiere incluir referencias a las librerías gráficas y la aplicación GUI podrá utilizar los métodos geométricos de la clase "Rectangulo-Geometrico".

Este principio es uno de los más simples a comparación de los otros principios, sin embargo el tenerlo de una manera correcta es dificultoso. Unir responsabilidades es algo que se hace a menudo pero encontrar y separar responsabilidades va más allá de lo que realmente es el diseño de software. [MARTIN&MARTIN: 2007]

2.5.2. Principio "Open/Closed"

Este principio plantea que una clase debe estar abierta a la extensión pero cerrada a la modificación.

Cuando se realiza un solo cambio a un programa resulta en cambios en cascada a los módulos dependientes el diseño cuenta con el síntoma de rigidez. El principio "Open/Closed" plantea que se realice la refactorización del sistemas de tal forma que este tipo de cambios no causen más modificaciones en todo el sistema. Si este principio se aplica correctamente, los nuevos cambios a ser realizados serán realizados adicionando código nuevo y no tocar código que ya era funcional. Este principio puede llevarse a cabo mediante el uso de interfaces o el uso de clases abstractas que plantean el uso de métodos abstractos.

Hacer uso de interfaces permiten que la clase dependiente de esta interfaz no sienta los cam-

bios realizados a las a las implementaciones de la interfaz. La Figura 11 muestra como la clase Cliente depende de la interfaz "Servidor" pero no sentirá los cambios realizados en las implementaciones de la interfaz "Servidor". Es decir la clase "Cliente" puede funcionar correctamente si trabaja con un "ServidorHTTP" o un "ServidorFTP".

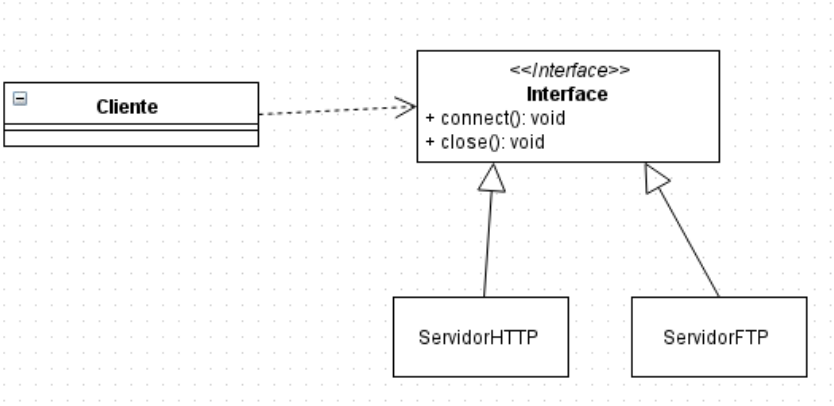

Figura 11: Interfaces para el principio Open/Closed

De muchas formas, el principio "Open/Closed" se encuentra en el corazón del diseño orientado a objetos. Estar conformes con este principio produce los mayores beneficios como reusabilidad, flexibilidad y mantenibilidad. El solo hecho de usar un lenguaje orientado a objetos no permite tener este principio dentro del diseño. Es necesario aplicar este principio a elementos que sufren cambios frecuentes. [MARTIN&MARTIN: 2007]

2.5.3. Principio "Liskov Substitution"

Este principio plantea que objetos deberían se reemplazables con instancias de sus subclases sin alterar el programa, es decir, si una clase depende de una implementación de alguna interfaz, esta debe tener la capacidad de utilizar cualquier implementación de la interfaz sin que sea necesario realizar cambios en el programa.

Los mecanismos principales detrás del principio "Open/Closed" son la abstracción, el polimorfismo y la herencia. Gracias a la herencia es posible crear clases que implementen métodos abstractos en sus clases base. Sin embargo no se sabe directamente si una jerarquía de clases es buena o mala para lograr el principio "Open/Closed" es por ello que el principio

“Liskov Substitution” es utilizado.

La importancia de este principio se convierte en obvia cuando se consideran las consecuencias de violar este principio. Un ejemplo es la existencia de un método *f* que toma como argumento la referencia de un objeto de la clase base B. Es posible que se pase como argumento D que es hija de B, pero esta D podría causar que *f* falle. Entonces D viola este principio. Es claro que D es frágil en la presencia de *f*.

Una solución parcial sería que los programadores arreglen el método *f* de tal forma que este funcione cuando reciba como argumento un objeto de tipo D. Sin embargo, esto viola el principio Open/Closed por que ahora *f* no se encuentra cerrada para todos las clases derivadas de la clase B.

Es por ello que un programa que requiera a cualquier elemento que herede de una clase B debe ejecutarse correctamente con cualquier clase hija de B.

El principio Open/Closed se encuentra en el corazón del diseño orientado a objetos. Cuando el principio Open/Closed se realiza, las aplicaciones gozan de mantenibilidad, reusabilidad y robustes. El principio “Liskov Substitution” es el principio principal que permite que el principio Open/Closed se lleve a cabo. La sustitubilidad de subclases permite que un módulo, expresado en términos de una clase base sea extensible sin la necesidad de modificación.

El término “es un” es muy amplio como para actuar como la definición de una subclase. La verdadera definición de una subclase es la sustitutibilidad, donde la sustitutibilidad se define tanto por un contrato implícito o explícito. [MARTIN&MARTIN: 2007]

2.5.4. Principio “Interface Segregation”

Este principio plantea que ninguna implementación de una interfaz debería ser forzada a depender de métodos que no usará.

Este principio se basa más en el diseño de una interfaz desde el punto que una interfaz no debería contener métodos que algunas implementaciones no requieran. En caso de ser así estos métodos pueden ser retirados a otra interfaz, ya que una clase puede implementar múltiples in-

terfaces este principio es aplicable en un diseño orientado a objetos.

Como ejemplo si una interfaz A cuenta con los métodos *f1*, *f2*, *f3*, *f4* y las clases B y C implementen los métodos *f1*, *f2*, *f3* y D *f1*, *f2*, *f3, f4* . Entonces es necesario sacar *f4* a otra interfaz A1 y el diseño quedaría que B, C implementan solo la interfaz A y D implementa las interfaces A y A1. La Figura 12 muestra el diseño después de aplicar el principio "Interface Segregation". [OTWELL: 2013]

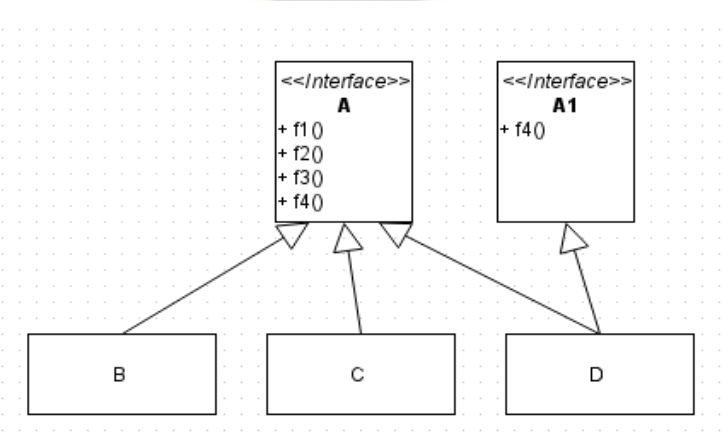

Figura 12: Diseño con el principio "Interface Segregation" aplicado

2.5.5. Principio "Dependency Inversion"

Este principio plantea que código de nivel superior (high-level) no debería depender de código de nivel inferior (low-level).

El código de nivel superior encapsula lógica y usa código de nivel inferior para funcionar pero este no debería depender directamente del código de nivel inferior. El código de nivel superior debería depender de una abstracción que se encuentra sobre el código de nivel inferior. El código de nivel inferior implementa operaciones básicas que le puede ser de utilidad a código de nivel superior. Este código debería estar en base a una abstracción.

La Figura 13 muestra un diseño en el que la clase "Autenticador" depende directamente de la clase "ProveedorUsuarioSql" y de la clase "Md5Hash".

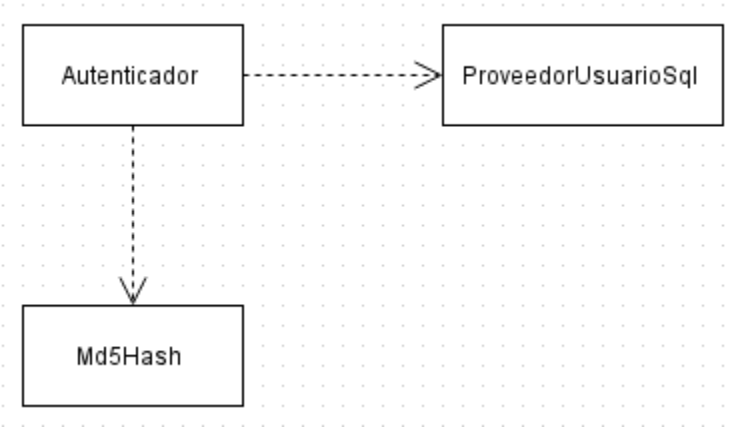

Figura 13: Dependencia de código de nivel inferior

Las clases “Md5Hash” y “ProveedorUsuarioSql” tienen dentro de ellas código de nivel inferior del cual la clase “Autenticador” de nivel superior depende. Para aplicar el principio “Dependency Inversion” es necesario que las clases de nivel inferior se encuentren bajo una abstracción y que las clases de nivel superior dependan de abstracciones. La Figura 14 muestra como se modifica el diseño de modo que se cumpla el principio “Dependency Inversion”.

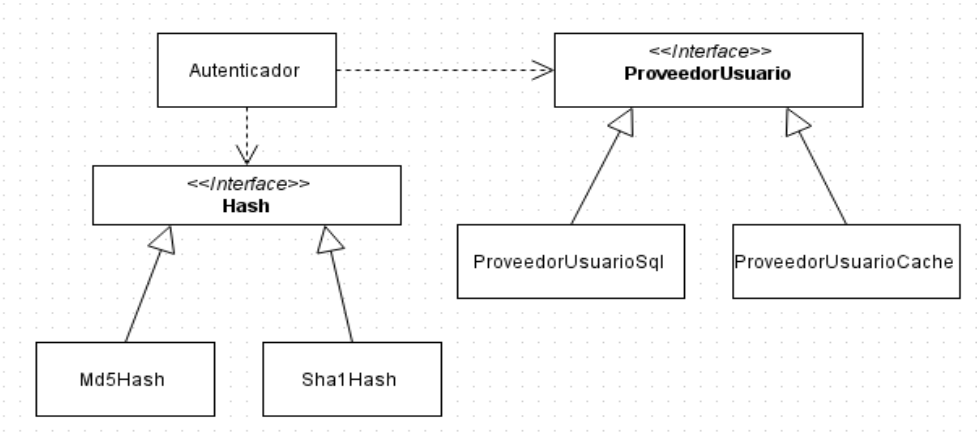

Figura 14: Diseño aplicando principio "Dependency Inversion"

De este modo la clase “Autenticador” de nivel superior solo depende de abstracciones y la implementación de estas abstracciones se encuentran en las clases de nivel inferior.

Es común que se creen estructuras de dependencia en las que clases de nivel superior donde se

guarda la lógica principal para que un módulo o aplicación funcione correctamente dependan de código de nivel inferior. De este modo, las clases de nivel superior se encuentran vulnerables a los cambios que se realicen a las clases de nivel inferior. Gracias a la programación orientada a objetos es posible invertir dicha dependencia de tal forma que las clases de nivel superior dependan de de abstracciones.

Un diseño que cuente con esta inversión de dependencia muestra un buen diseño orientado a objetos. La aplicación correcta de este principio es necesaria para la creación de frameworks reutilizables. Tanto abstracciones y detalles se mantienen aisladas una de otra, es por ello que el código es más fácil de mantener. [MARTIN&MARTIN: 2007]

2.6. Metodologías y Modelos de Desarrollo

Una buena práctica de organización de proyectos es hacer uso de una metodología de desarrollo para poder así organizar al equipo, los tiempos y recursos. En la actualidad por el gran nivel competente entre empresas es necesario producir software de la manera más rápida posible, de esta forma es posible obtener una retro alimentación de los usuarios para poder realizar mejoras. En la actualidad, las diferentes metodologías ágiles ganan popularidad entre empresas al permitir la entrega constante de productos funcionales.

2.6.1. Scrum

Desarrollada por Ken Schwaber y Jeff Shuterland en los años 90 Scrum es un marco de trabajo utilizado para la construcción de productos complejos. Scrum al no ser un proceso o técnica, pero este cuenta con diferentes procesos y técnicas. Scrum muestra de forma clara la relación eficaz entre la administración de un producto y las buenas prácticas de desarrollo. [SCHWABER&SUTHERLAND: 2013]

Scrum es un marco de trabajo con el cual es posible trabajar con problemas complejos con los que lidian diferentes equipos haciendo entrega de productos de valor alto de una forma productiva y creativa.

Scrum al ser un marco de trabajo ágil, sigue los principios planteados en le manifiesto ágil. La

Tabla 11 muestra muestra estos principios. [BECK y OTROS: 2001]

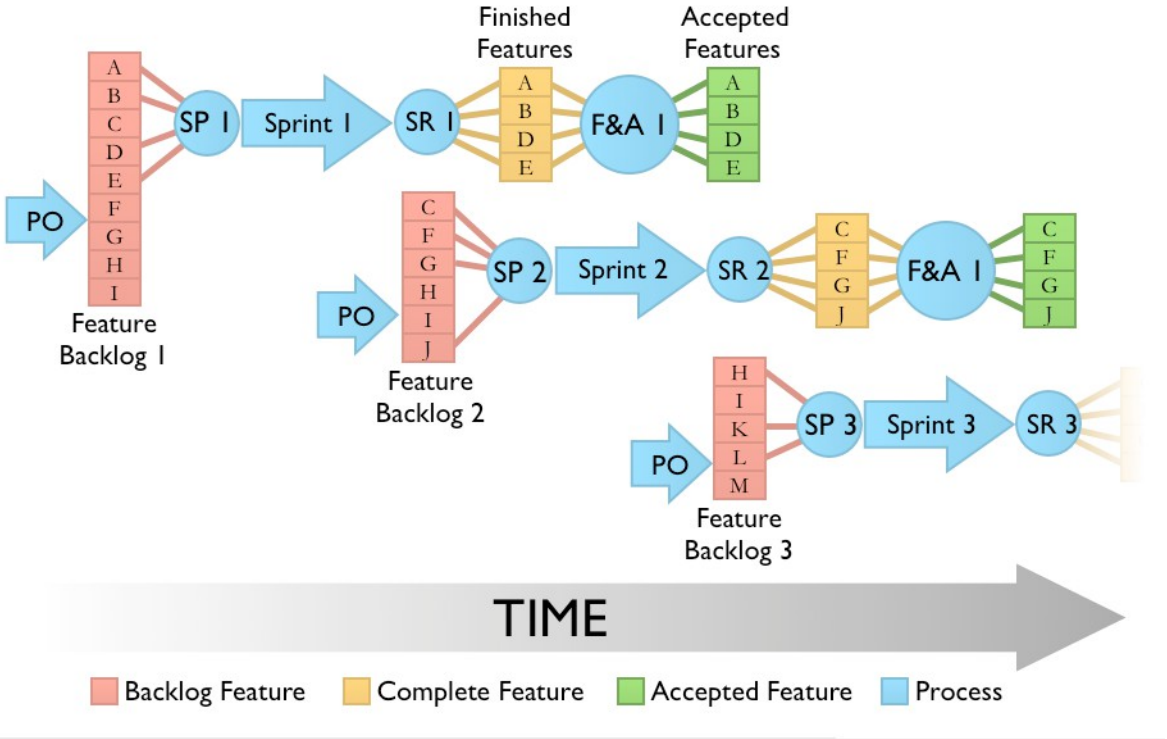

Figura 15: Flujo de trabajo en Scrum

#	**Principio**
1	Satisfacer al cliente mediante la entrega temprana y continua de software con valor
2	Es aceptable que los requisitos cambien, incluso en etapas tardías de desarrollo. Los procesos agiles aprovechan el cambio para proporcionar una ventaja competitiva al cliente.
3	Entrega frecuente de software funcional, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible
4	Los responsables de negocio y desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto
5	Los proyectos se desarrollan en torno a individuos motivados. Es necesario brindarles el entorno y apoyo que estos requieren así confiarles la ejecución del trabajo

6	El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara
7	El software funcional es la medida principal de progreso
8	Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida
9	La atención continua a la excelencia técnica y al buen diseño mejora la agilidad
10	La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado es esencial
11	Las mejores arquitecturas, requisitos y diseños surgen de equipos auto-organizados
12	A intervalos regulares el equipo reflexiona sobre como ser más efectivo para poder así ajustar y perfeccionar su comportamiento en consecuencia

Tabla 11: Principios del manifiesto ágil

Scrum implica hacer uso de los diferentes eventos, roles y artefactos que este proporciona. La Figura 15 muestra el flujo de trabajo de un proyecto que aplica Scrum siendo cada fila una iteración de esta metodología.

2.6.1.1. Roles de Scrum

Los equipos que hacen uso de Scrum hacen la entrega de productos funcionales de forma iterativa e incremental para poder así maximizar las oportunidad de retro alimentación. Las entregas incrementales de productos terminados garantiza una versión potencialmente útil del producto. Un equipo Scrum son auto organizados y funcionales. Los miembros de un equipo auto organizado eligen las mejores opciones para poder realizar el trabajo, y no así, ser dirigidos por otros miembros fuera del equipo. Un equipo funcional tiene todos los elementos necesarios para poder realizar el trabajo sin la necesidad de depender de elementos exteriores al equipo. Un equipo de Scrum consiste de un “Product Owner”, Equipo de Desarrollo y un “Scrum Master”. La Tabla 12 muestra la definición y función que cumple cada rol.

Rol	**Función del Rol**
Product Owner	Este miembro es el responsable de maximizar el valor del producto y el

	trabajo del Equipo de Desarrollo. El Product Owner es el único miembro responsable del backlog del producto. Por lo general este miembro es una sola persona y no un comité, sin embargo, este se encarga de plasmar los requerimientos de algún comité en el backlog del producto. Las decisiones realizadas por el Product Owner se hacen visibles en el backlog del producto. De esta forma, nadie más que el Product Owner puede ordenar que el Equipo de Desarrollo trabaje en otro tipo de requerimientos, y el Equipo de desarrollo no tiene permitido en trabajar en base a que otros digan.
Equipo de Desarrollo	El Equipo de Desarrollo consiste de profesionales que realizan el trabajo de realizar entregas de un producto hecho al final de cada sprint (iteración). Un Equipo de Desarrollo es auto-organizado y funcional para poder así maximizar la efectividad del equipo. Por lo general un equipo de desarrollo debe estar formado entre tres y nueve elementos, de esta forma no tener conflictos de organización ni minimizar los resultados en cada sprint.
Scrum Master	El Scrum Master es responsable de asegurar que todas las actividades de Scrum se lleven de forma correcta. Es necesario que este rol muestre y haga entender la teoría, reglas y prácticas de Scrum. Es necesario que el Scrum Master ayude a los elementos fuera del equipo a entender cuales son las interacciones con el equipo son útiles y cuales no, de este modo el Scrum Master ayudará a modificar y adaptar estas interacciones para poder maximizar el valor de todo el equipo en conjunto.

Tabla 12: Roles de Scrum

2.6.1.2. Eventos de Scrum

Los eventos que Scrum provee son utilizados para regularizar y minimizar la necesidad de reuniones. Todos estos eventos se encuentran relacionas con el tiempo, es decir, cada evento tiene un máximo de tiempo pre establecido. El sprint al ser el evento más importante además de contener a los otros eventos, es una oportunidad formal de inspeccionar y adaptar los diferen-

tes puntos de mejora que un equipo tenga. El objetivo principal de estos eventos es el de permitir una transparencia crítica y el no incluir alguno de estos eventos podría reducir la transparencia que se desea. Los eventos que Scrum propone son el Sprint, Planificación de Sprint, Daily Scrum, Revisión del Sprint y la Retrospectiva del Sprint. La Tabla 13 muestra las actividades que se realizan en cada evento.

Evento	**Actividades**
Sprint	El corazón de Scrum se yace en el Sprint o iteración, la cual cuenta con un lapso de tiempo menor o igual a un mes en el cual es necesario presentar una versión incrementada, utilizable, con el potencial de ser el producto final en base a una meta inicial. Un Sprint es consistente por los esfuerzos de desarrollo empleados. Un Sprint inicia al finalizar un Sprint anterior. Un Sprint consiste de la Planificación de Sprint, Daily Scrum, Revisión del Sprint y la Retrospectiva del Sprint.
Planificación de Sprint	Este evento realiza la planificación de todas las metas a cumplirse durante el Sprint, la duración de esta planificación debe ser menor o igual a ocho horas, sin embargo, la duración de la planificación es directamente proporcional a la duración del Sprint, es decir, la planificación de un Sprint corto debe ser corta. En este evento el Scrum Master debe encargarse de que el equipo entienda el propósito del Sprint a realizarse. La meta del Sprint es un conjunto de objetivos que pueden hacerse gracias al backlog del producto. La meta es una guía para el equipo de desarrollo para la auto-organización de este. Estos objetivos son un conjunto de historias de usuario a desarrollarse que se obtienen del backlog del producto a las cuales se asignan tareas, puntos de complejidad, encargados que realizaran la historia de usuario.
Daily Scrum	Es una reunión que se realiza todos los días con una duración máxima de quince minutos en la que todos los miembros del equipo tocan los siguientes puntos:

	• Que se realizó el día anterior • Que se realizará en la jornada presente • La existencia de algún elemento que bloquee al miembro del equipo
Revisión del Sprint	Este evento se realiza al haber finalizado el Sprint para poder inspeccionar las mejoras en el producto trabajado y verificar los cambios que se realizaron al backlog del producto. Para un Sprint de un mes se recomienda que este evento dure a lo mucho cuatro horas. Los diferentes miembros del equipo comparten los avances que se realizaron durante el Sprint con las partes interesadas en el proyecto, por lo general clientes que son invitados previamente por el Product Owner. El Product Owner debe los elementos del backlog del producto que fueron realizados, basados en eso, el Equipo de Desarrollo demuestra el trabajo que realizó y responde a las preguntas respecto al trabajo realizado en el Sprint actual. Es posible que el mercado al que apunta la aplicación haya cambiado, es por ello que en este evento es necesario aclarar este tipo de tópicos para poder aumentar el valor de la siguiente iteración. En general, el grupo completo colabora de tal forma que se consiguen puntos a tocar en la planificación del siguiente Sprint que puedan aumentar el valor del producto.
Retrospectiva del Sprint	Este evento permite que el equipo encuentre falencias o propuestas para mejorar el siguiente Sprint. Este evento se lleva a cabo tras haber concluido la revisión del Sprint y antes de comenzar la planificación del siguiente Sprint. Se recomienda que la duración de este evento sea a lo mucho de tres horas para sprints de un mes. Durante este evento el equipo debe planificar diferentes formas para poder incrementar la calidad del producto. Para ello es necesario inspeccionar como fueron las personas del equipo, relaciones y uso de las herramientas de tal forma que sea posible identificar falencias o mejoras. De este modo, al finalizar la Retrospectiva del Sprint el equipo habrá encontrado mejoras que puedan ser aplicadas al siguiente Sprint. No es obligatorio que estas mejoras

	sean encontradas y aplicadas durante y después de la Retrospectiva del Sprint, estas mejoras pueden ser aplicadas en cualquier momento de un Sprint.

Tabla 13: Eventos de Scrum

2.6.1.3. Artefactos de Scrum

Los artefactos que Scrum utiliza se encuentran específicamente diseñados par maximizar la transparencia de información clave de tal forma que todo el equipo entienda el artefacto de la misma forma. Los artefactos que Scrum utiliza son el Backlog del Producto, Backlog del Sprint y el Incremento. La Tabla 14 muestra las funcionalidades de cada artefacto.

Artefacto	**Descripción**
Backlog del Producto	El Backlog del Producto es una lista los requerimientos, funciones, características, mejoras y arreglos necesarios para el producto, además de ser la única fuente de requerimientos al cual pueden realizarse cambios. El Product Owner es el encargado de mantener y gestionar este artefacto incluyendo su contenido, disponibilidad y orden. Al ser Scrum una metodología iterativa, el backlog del producto nunca se encontrará finalizada, mientras exista el producto existirá el backlog del producto. Inicialmente el contenido de este artefacto solo muestra los requerimientos que fueron comprendidos. Este artefacto evoluciona a medida que el producto y el entorno en el que el producto funcionará evolucionan, es decir, cualquier cambio en requerimientos de negocio o condiciones de mercado puede causar cambios en el backlog del producto. Es necesario refinar el backlog del producto adicionando estimados, detalles y orden a los elementos que yacen en este. Durante este proceso en el que trabajan el Product Owner y el Equipo de desarrolla los elementos del backlog del producto son revisados.
Backlog del Sprint	El backlog del sprint es un conjunto de elementos del backlog del producto que son elegidos para llevarse a cabo durante un Sprint, además de

	tener un plan para entregar la mejora del producto y cumplir la meta del Sprint. El backlog del sprint es un pronóstico del equipo de desarrollo sobre que funcionalidad tendrá la siguiente mejora del producto y el trabajo necesario para poder finalizar dichas funcionalidades. Este artefacto hace visible todo el trabajo necesario identificado por el Equipo de Desarrollo para cumplir la meta del Sprint. A medida que más trabajo es necesario, es necesario adicionarlo al backlog del sprint. Por otro lado, a medida que se realice el trabajo o se lo complete, el trabajo restante estimado se actualiza. A diferencia del backlog del producto, el backlog del sprint solo es administrado por el Equipo de Desarrollo, de tal forma que solo este equipo puede realizar cambios en este artefacto, adicionar elementos necesarios o remover elementos innecesarios.
Incremento o mejora del producto	Al final un sprint el incremento o mejora del producto es el total de todos los elementos completados que pertenecen al backlog del sprint y el valor de las mejoras del producto de sprints anteriores. Al finalizar un sprint un nuevo incremento esta hecho lo que significa que este incremento debe encontrarse en una condición útil independientemente que el Producto Owner decida que el incremento se publicará o no.

Tabla 14: Artefactos de Scrum

2.6.2. El modelo Open-Source

Open-Source es un modelo de desarrollo el cual permite (a través de una licencia Open-Source) acceso a los planos, diseño o fuente de un producto, además de la distribución de estos incluyendo mejoras posteriores hechas por otras personas. El hecho de abrir el código fuente abre las puertas a la creación de diferentes modelos de producción y comunidades interactivas.

Las comunidades juegan un papel importante en este modelo de desarrollo ya que los miembros, al ser estas participes en el desarrollo, documentación, reparación, traducción y prueba de un producto, generan una retro alimentación que permite validar los requerimientos del pro-

ducto lo cual beneficia a futuras versiones de este. A diferencia de un modelo clásico de desarrollo de Software Libre en el que las fuentes de un proyecto son liberadas hasta que un proyecto se encuentre en una fase beta, el modelo Open-Source plantea el desarrollo de un producto a través de Internet en el que las fuentes, documentación y diseños de un producto siempre se encuentren públicas, de este modo tareas como documentar, reparar errores, depurar y otras se agilizan. [RAYMOND: 2000]

La existencia de varios productos Open-Source genera la existencia de diferentes comunidades dispersas por gente de todo el mundo, con modelos de producción y organización distintos lo cual hace complicado la aplicación de una metodología de desarrollo específica. Eric S. Raymond en su ensayo "La Catedral y el Bazar" reconoce un conjunto de ideas que posteriormente llegan a formar los principios del modelo Open-Source. La muestra los principios más importantes de este modelo. [RAYMOND: 2000], [DIXON: 2007]

Principio	**Descripción**
Transparencia	Hace referencia a que cualquier persona puede ver el código fuente del proyecto, los cambios que se realizaron a través del tiempo, documentación y planes para futuras versiones del producto
Participación abierta	Aceptar la participación de otras personas, aceptando o considerando contribuciones en código, documentación traducción, observaciones y otros
Entregas tempranas y continuas	No solo hace referencia a las entregas del producto como tal, este principio incluye la entrega temprana y continua de otros elementos como diseños, características de futuras versiones y otros utilizando hitos para delimitarlas. De este modo es posible recibir una retro alimentación de la comunidad
Modularidad	Este principio propone mantener los límites bien definidos entre los diferentes componentes de un producto

Tabla 15: Principios del modelo Open-Source

2.6.3. Open Scrum

Scrum al ser una metodología de desarrollo ágil cumple los principios propuestos en el mani-

fiesto ágil (ver sección 2.6.1.). Los principios propuestos en le modelo Open-Source con los principios propuestos en el manifiesto ágil comparten similitudes. [DIXON: 2007]

- Tanto Open-Source como Scrum comparten los principios de transparencia, participación abierta (en el caso de Scrum abierta a todo el equipo) y entregas tempranas y continuas.
- Tanto Scrum y Open-Source usan prácticas como la creación hitos en las versiones del producto
- Los Equipos de Desarrollo en Scrum generalmente son pequeños. En proyectos Open-Source el equipo principal de desarrolladores por lo general son pequeños.
- Tanto en Scrum y en proyectos Open-Source los equipos son auto-organizados.

La metodología Open Scrum propuesta por James Dixon, plantea una modificación conceptual de algunos roles, eventos y artefactos de la metodología Scrum de tal forma que sea posible hacer uso de esta metodología en proyectos Open-Source.

La Tabla 16 muestra la modificación conceptual de algunos elementos de Scrum en Open Scrum.

Característica en Scrum	**Característica en Open Scrum**
Backlog del Producto La lista completa de características (historias de usuario) para el producto	Backlog del Producto Este artefacto debe estar disponible públicamente por Internet (blogs, sitio oficial del proyectos, wiki y otros). Ya que contribuciones son aceptadas desde la comunidad el backlog del Sprint debe ser actualizado.
Planificación del Sprint Una reunión en la que se eligen algunos elementos del backlog del producto para ser realizados durante el sprint	Planificación del Sprint Los detalles de estas reuniones deben estar disponible públicamente por Internet (blogs, sitio oficial del proyectos, wiki y otros). Los asistentes y la forma en como se realizará la reunión esta en base a como lo decidan los lideres del proyecto.

Retrospectiva del Sprint Una reunión en la que se discuten las nuevas características realizadas durante el sprint y que lecciones se aprendieron.	Retrospectiva del Sprint Los detalles de estas reuniones deben estar disponible públicamente por Internet (blogs, sitio oficial del proyectos, wiki y otros). Es posible adjuntar a esto capturas de pantalla del avance realizado.
Product Owner La persona responsable del backlog del producto	Product Owner En el caso de los proyectos Open-Source, el creador del proyecto, administrador o miembros del equipo principal de desarrollo son los posibles candidatos para tomar este rol. En el caso de un proyecto Open-Source comercial este rol debe ser tomado por el Gerente del Product
Scrum Master La persona responsable de hacer seguimiento y verificar que los eventos de Scrum se cumplan	Scrum Master En varios proyectos este rol estará a cargo del creador del proyecto, administrador o miembros del equipo principal de desarrollo.
Equipo de Desarrollo El equipo de desarrolladores, arquitectos, diseñadores, testers y documentadores que realizaran el producto	Equipo de Desarrollo Al ser un proyecto Open-Source, existirán contribuciones por parte de la comunidad (documentación, código, traducciones, reportes de fallas). Al ser necesaria una diferencia, es posible dividir el equipo de desarrollo en el equipo principal de desarrollo y el equipo extendido de desarrollo.
Sprint Un evento de hasta 4 semanas en el que se deben terminar características (probadas y utilizables)	Sprint La duración de un Sprint podría alargarse hasta doce semanas y que el Sprint puede ser llevado a cabo por una persona o desarrolladores de medio tiempo.
Daily Scrum Una reunión diaria con un máximo de 15 minutos en la que cada miembro da a conocer lo que hizo el	Daily Scrum El equipo principal de desarrolladores necesita estar en contacto frecuentemente y los resultados de estas interacciones debe estar disponibles públicamente. En el

día anterior, lo que hará ese día y si existen obstáculos presentes.	caso de proyectos Open-Source comerciales con muchos desarrolladores las reuniones deben realizarse a diario. En caso de un desarrollador o desarrolladores de medio tiempo, estas reuniones deberían celebrarse cada una o dos semanas.

Tabla 16: Cambios conceptuales Scrum y Open Scrum

Tras los cambios conceptuales el flujo de trabajo se mantiene adicionando un nuevo elemento a este que consta de la comunidad que es muy importante dentro de un proyecto Open-Source. La Error: no se encontró el origen de la referencia muestra el flujo de trabajo aplicando Open Scrum, de la misma forma que la metodología Scrum se realizan todos los eventos adicionando un nuevo elemento que es la comunidad.

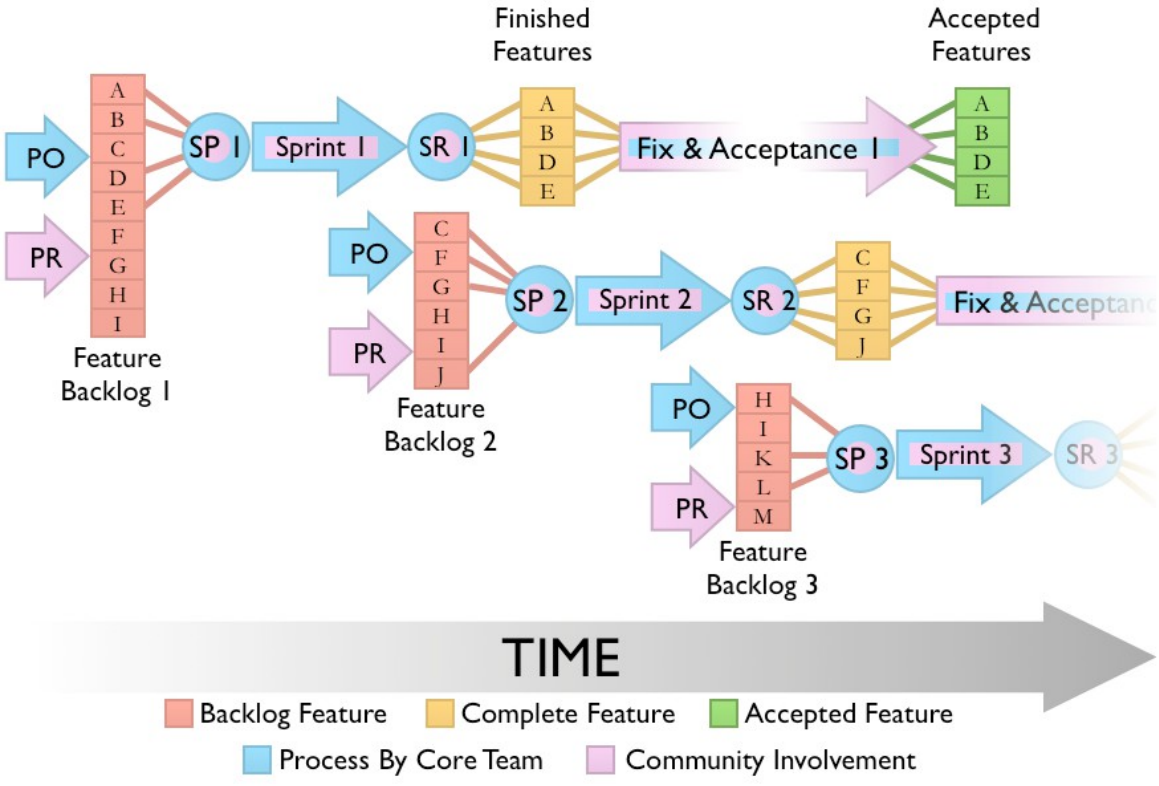

Figura 16: Flujo de trabajo de Open Scrum

CAPÍTULO 3

MARCO APLICATIVO

El desarrollo de la plataforma Beehive fue desarrollada haciendo uso de la metodología de desarrollo Open Scrum (Ver Sección 2.6.3). El presente capítulo muestra la aplicación de esta metodología en sus diferentes etapas para el desarrollo del prototipo. Para el control de Backlog, Historias de Usuario, Tareas y Sprints se hace uso de la herramienta Open-Source Taiga[16].

Inicialmente es necesario plantear un conjunto de definiciones las cuales se utilizan a lo largo de la etapa de desarrollo y pruebas. Estas definiciones permiten brindar una mejor comprensión de este capítulo. La Tabla 17 muestra las definiciones planteadas.

Término	**Definición**
Template	Se define como el conjunto de abstracciones de un modelo específico de hardware. Ejemplo: un ventilador de la marca "Ejemplo-M16" cuenta con atributos funcionales como la velocidad, estado y otros además de acciones como encender, apagar y cambiar velocidad. Estos atributos y acciones se abstraen en un Template.
Data Stream	Dependiente de un Template específico un Data Stream se define como la abstracción de atributos funcionales de un Template que cambiarán con el tiempo y podrán ser monitoreados y almacenados en la plataforma Beehive a medida que estos atributos cambien. Ejemplo: los atributos de velocidad y estado de un ventilador pueden cambiar con el tiempo, es por eso que un Template para un ventilador podría tener dos Data Streams, uno para la velocidad y otro para el estado.
Comando	Dependiente de un Template específico un Comando se define como la abstracciones de acciones de un Template que puede ser ejecutado remotamente a través de la plataforma Beehive y que esta acción llegará a un

16 Proyecto Beehive en Taiga.io - https://tree.taiga.io/project/donkeysharp-beehive-platform

	dispositivo conectado. Un comando puede contar con cero o más argumentos. Ejemplo: las acciones que se pueden realizar remotamente en un ventilador podrían ser encender, apagar, cambiar la velocidad, es por eso que un Template para un ventilador podría tener tres Comandos, para el encendido, apagado y cambio de velocidad con el argumento de velocidad.
Dispositivo	Se define como una instancia única de un Template, la unicidad de este Dispositivo tendrá los Data Streams y Comandos definidos en el Template al que este pertenece. La unicidad de estos dispositivos es gracias a la generación de llaves únicas UUID en los campos MQTT-PUB Key y MQTT-SUB Key. Ejemplo: se compra dos ventiladores del modelo "Ejemplo-M16" ambos tendrán los atributos como velocidad y estado además de las acciones como encender, apagar y cambiar velocidad. Cada ventilador puede ser utilizado independientemente de otros ya que estas son instancias únicas de un Template que abstrae las funcionalidades de un ventilador modelo "Ejemplo-M16"
Canales de Comunicación	Los canales de comunicación es el mecanismo que brinda la plataforma Beehive para que Dispositivos de hardware puedan comunicarse con la plataforma. De esta forma el dispositivo podrá enviar los diferentes valores de sus atributos a través de estos canales de comunicación además de recibir diferentes comandos realizados por un usuario.
Pertenencia	Uno o más usuarios pueden poseer un dispositivo de hardware específico lo cual permite a estos usuarios poder visualizar atributos o ejecutar acciones sobre este.

Tabla 17: Definiciones previas para la etapa de desarrollo

3.1. Backlog Inicial de Historias de Usuario

La Tabla 18 muestra el backlog inicial de historias de usuario para la plataforma Beehive en orden de importancia usando como identificadores los asignados por la aplicación Taiga. El nombre de las historias de usuario que pertenecen al backlog del producto son las mismas

historias de usuario identificadas (ver sección 3.2.).

ID	Nombre de Historia de Usuario
#13	Identificación de tecnologías
#12	Diseño inicial del modelo de datos
#14	Diseño inicial de pantallas de la plataforma Web
#3	Administración de Templates
#6	Administración de Dispositivos
#4	Administración de Comandos y Argumentos
#9	Implementación de los Canales de Comunicación
#33	Seguridad en los Canales de Comunicación
#37	Integración entre los Canales de Comunicación y la Plataforma Web
#5	Administración de Data Streams para un Template
#8	Panel de visualización de Data Streams de un Dispositivo
#7	Panel de Comandos para un Dispositivo
#10	Librerías para dispositivos de hardware: Arduino
#35	Librerías para dispositivos de hardware: Raspberry Pi
#11	Servicios RESTful para autenticación
#58	Servicios RESTful para Dispositivos
#59	Servicios RESTful para Templates
#60	Servicios RESTful para Comandos y Argumentos
#61	Servicios RESTful para Data Streams

Tabla 18: Backlog inicial de historias de usuario

3.2. Descripción de Historias de Usuario identificadas

La arquitectura de Beehive cuenta principalmente con cuatro elementos (ver Figura 1: Arquitectura de la plataforma Beehive). Teniendo esta arquitectura como base se definen cuatro categorías identificados se ven en la Tabla 19.

Nombre de Categoría
Plataforma Web
Canales de Comunicación
Dispositivos de Hardware
RESTful API

Tabla 19: Categorías Identificados

Para cumplir los objetivos planteados (ver capítulo 1) es necesario tener un conjunto de historias de usuario para tener en claro las características que se desean realizar. En un inicio se identifican 19 historias de usuario divididas en las categorías identificadas, desde la Tabla 20 hasta la Tabla 38 se muestran las historias de usuario identificadas.

		Historia de Usuario
ID: #13	**Nombre:** Diseño inicial del modelo de datos	
Puntos: 8	**Usuario:** Equipo Desarrollo	**Categoría:** Ninguno
Descripción: Diseño inicial del modelo de datos tomando en cuenta: Dispositivos, Templates y Comandos/Argumentos		

Tabla 20: H.U. Diseño inicial del modelo de datos

		Historia de Usuario
ID: #12	**Nombre:** Diseño inicial de pantallas de la plataforma Web	
Puntos: 5	**Usuario:** Equipo Desarrollo	**Categoría:** Ninguna
Descripción: Diseño inicial de pantallas de la plataforma Web. Pantallas de perfil e inicio de sesión		

Tabla 21: H.U. Diseño inicial de pantallas de la plataforma Web

		Historia de Usuario
ID: #14	**Nombre:** Identificación de tecnologías	
Puntos: 13	**Usuario:** Equipo Desarrollo	**Categoría:** Ninguna
Descripción: Identificación de tecnologías. Websockets con SockJS y MQTT con Mosquitto		

Tabla 22: H.U. Identificación de tecnologías

		Historia de Usuario
ID: #3	**Nombre:** Administración de Templates	
Puntos: 5	**Usuario:** Usuario Web	**Categoría:** Plataforma Web
Descripción: Como usuario, deseo administrar templates de tal forma que pueda asociar dispositivos a un template. Un template es la abstracción del modelo de dispositivo, entonces cada dispositivo que use un template tendrá las mismas propiedades del template.		

Tabla 23: H.U. Administración de Templates

		Historia de Usuario
ID: #6	**Nombre:** Administración de Dispositivos	
Puntos: 3	**Usuario:** Usuario Web	**Categoría:** Plataforma Web
Descripción: Como usuario, deseo administrar dispositivos de tal forma que estos puedan ser asociados un template, además de tener llaves únicas para poder acceder a la plataforma desde dispositivos de hardware.		

Tabla 24: H.U. Administración de Dispositivos

		Historia de Usuario
ID: #4	**Nombre:** Administración de Comandos y Argumentos	
Puntos: 3	**Usuario:** Usuario Web	**Categoría:** Plataforma Web
Descripción: Como usuario, deseo administrar en una forma genérica comandos y argumentos de tal forma estos puedan ser asociados a un template.		

Tabla 25: H.U. Administración de Comandos y Argumentos

		Historia de Usuario
ID: #9	**Nombre:** Implementación de los Canales de Comunicación	
Puntos: 5	**Usuario:** Usuario Web	**Categoría:** Canales de Comunicación
Descripción: Como usuario, deseo un mecanismo para manejar los canales de comunicación entre dispositivos de hardware y la plataforma Beehive de tal forma que esta comunicación sea realizada a través del protocolo MQTT		

Tabla 26: H.U. Implementación de los Canales de Comunicación

		Historia de Usuario
ID: #33	**Nombre:** Seguridad en los Canales de Comunicación	
Puntos: X	**Usuario:** Usuario Web	**Categoría:** Canales de Comunicación
Descripción: Como usuario, deseo que cada dispositivo conectado a la plataforma Beehive vía MQTT solo pueda publicar o suscribirse a los tópicos en los cuales tenga permiso de tal forma que otros dispositivos no interfieran con la comunicación de otros dispositivos		

Tabla 27: H.U. Seguridad en los Canales de Comunicación

		Historia de Usuario
ID: #37	**Nombre:** Integración de Canales de Comunicación con la plataforma web	
Puntos: 8	**Usuario:** Usuario Web	**Categoría:** Plataforma Web
Descripción: Como usuario, deseo integrar los canales de comunicación con la plataforma web a través de WebSockets		

Tabla 28: H.U. Integración de Canales de Comunicación con la plataforma web

		Historia de Usuario
ID: #8	**Nombre:** Panel de visualización de Data Streams para un Dispositivo	
Puntos: 5	**Usuario:** Usuario Web	**Categoría:** Plataforma Web
Descripción: Como usuario, deseo tener un panel que visualice todos los Data Streams asociados a un dispositivo de tal forma que los datos se visualicen en gráficas estadísticas en tiempo real		

Tabla 29: H.U. Panel de visualización de Data Streams para un Dispositivo

		Historia de Usuario
Número: #5	**Nombre:** Administración de los Data Streams para un Template	
Puntos: 3	**Usuario:** Usuario Web	**Categoría:** Plataforma Web
Descripción: Cada dispositivo puede enviar varios tipos de datos a través de Data Streams. Como usuario, deseo administrar los Data Streams de tal forma que pueda declarar la unidad de medidas y el tipo de gráfica estadística en el cual los datos serán visualizados y estos estén asociados a un Template		

Tabla 30: H.U. Administración de los Data Streams para un Template

		Historia de Usuario
Número: #7	**Nombre:** Panel de Comandos para un Dispositvo	
Puntos: 3	**Usuario:** Usuario Web	**Categoría:** Plataforma Web
Descripción: Como usuario, deso tener un panel en el cual, dado un dispositivo, pueda ejecutar comandos asociados a este de tal forma que estos sean enviados a los dispositivos de hardware		

Tabla 31: H.U. Panel de Comandos para un Dispositvo

		Historia de Usuario
Número: #10	**Nombre:** Librerias para Dispositivos Arduino	
Puntos: 3	**Usuario:** Desarrollador Arduino	**Categoría:** Dispositivos de Hardware
Descripción: Como un desarrollador de Arduino, deseo una librería que pueda comunicarse con la plataforma Beehive, de tal forma que pueda enviar a y recibir datos a través de los canales de comunicación.		

Tabla 32: H.U. Librerias para Dispositivos Arduino

		Historia de Usuario
Número: #35	**Nombre:** Librerías para dispositivos Raspberry Pi	
Puntos: 3	**Usuario:** Desarrollador Raspberry Pi	**Categoría:** Dispositivos de Hardware
Descripción: Como un desarrollador de Raspberry Pi, deseo una librería que pueda comunicarse con la plataforma Beehive, de tal forma que pueda enviar a y recibir datos a través de los canales de comunicación.		

Tabla 33: H.U. Librerías para dispositivos Raspberry Pi

		Historia de Usuario
Número: #11	**Nombre:** Servicios RESTful para Autenticación	
Puntos: 3	**Usuario:** Desarrollador	**Categoría:** RESTful API
Descripción: Como desarrollador, deseo que la RESTful API cuente con servicios para autenticación de tal forma que cualquier usuario pueda autenticarse a través de estos servicios		

Tabla 34: H.U. Servicios RESTful para Autenticación

Historia de Usuario		
Número: #58	**Nombre:** Servicios RESTful para Dispositivos	
Puntos: 3	**Usuario:** Desarrollador	**Categoría:** RESTful API
Descripción: Como desarrollador deseo servicios RESTful para dispositivos de tal forma que pueda obtener y realizar las acciones de un dispositivo		

Tabla 35: H.U. Servicios RESTful para Dispositivos

Historia de Usuario		
Número: #59	**Nombre:** Servicios RESTful para templates	
Puntos: 3	**Usuario:** Usuario Web	**Categoría:** RESTful API
Descripción: Como desarrollador deseo servicios RESTful para los templates de tal forma que pueda obtener información de templates y realizar acciones en templates		

Tabla 36: H.U. Servicios RESTful para templates

Historia de Usuario		
Número: #60	**Nombre:** Servicios RESTful para comandos y argumentos	
Puntos: 3	**Usuario:** Desarrollador	**Categoría:** RESTful API
Descripción: Como desarrollador, deseo leer, modificar, adicionar y eliminar mediante comandos y argumentos de tal forma que estas acciones se realicen a través de un servicio RESTful		

Tabla 37: H.U. Servicios RESTful para comandos y argumentos

Historia de Usuario		
Número: #61	**Nombre:** Servicios RESTful para Data Streams	
Puntos: 3	**Usuario:** Desarrollador	**Categoría:** RESTful API
Descripción: Como desarrollador, deseo adicionar, eliminar y obtener la estructura de los Data Streams de tal forma que estas acciones se realicen a través de un servicio RESTful		

Tabla 38: H.U. Servicios RESTful para Data Streams

3.3. Sprints

Desde el inicio del desarrollo en Octubre de 2014 se realizan siete sprints completando así los

alcances propuestos (ver sección 1.6.2.). Cada sprint cuenta con su backlog detallado con las tareas a realizar para cada historia de usuario, las casos de prueba para verificar la funcionalidad del programa y finalmente el incremento funcional tras haber finalizado el sprint.

3.3.1. Sprint 0 – Reconocimiento

Este sprint define un esquema inicial como la estructura base del proyecto, definición inicial de pantallas, modelo inicial de datos, adecuación con las herramientas y tecnologías con las cuales el prototipo se desarrolla.

Fecha Inicio: 06 OCT 2014

Fecha Finalización: 31 OCT 2014

Sprint Backlog

La Tabla 39 muestra el backlog del Sprint 0 junto con las tareas por realizar en cada historia de usuario.

ID	Nombre de Historia de Usuario
#13	Identificación de tecnologías
Tareas • Identificar y evaluar tecnologías para el backend • Identificar y evaluar tecnologías para el frontend • Crear el proyecto base con la capacidad de autenticación	
#12	Diseño inicial del modelo de datos
Tareas • Definir el modelo de datos inicial	
#14	Diseño inicial de pantallas de la plataforma Web
Tareas • Crear bosquejos del posible diseño de la plataforma web • Plasmar los bosquejos en html, css y javascript	

Tabla 39: Backlog Sprint 0

La Tabla 40 muestra los casos de prueba realizados en este sprint.

Detalle		
Debería ser posible iniciar y terminar sesión en la plataforma Beehive		
Precondiciones		
• No tener un usuario que haya iniciado sesión		
#	**Pasos a realizar**	**Resultados Esperados**
1	Entrar a la pantalla de inicio de sesión	Visualizar los campos de usuario y contraseña
2	Introducir las credenciales de un usuario válido	Haber sido redirigido a la pantalla de inicio de la plataforma
3	En el menú ubicado en la parte superior derecha hacer clic en el link “logout” para finalizar sesión	Haber sido redirigido a la pantalla de presentación de la plataforma
Detalle		
Debería ser posible visualizar y editar los datos del perfil de un usuario		
Precondiciones		
• Haber iniciado sesión con un usuario válido		
#	**Pasos a realizar**	**Resultados Esperados**
1	Ingresar a la pantalla de perfil haciendo clic en el link “Profile” ubicado en el menú de la parte superior derecha	Visualizar los detalles del usuario (imagen de perfil, nombre, username, país y correo electrónico)
2	Hacer clic en el botón editar ubicado en la parte superior derecha de los detalles de perfil	Ser redirigido a la pantalla de edición de perfil
3	Llenar los diferentes campos del panel “Basic Information” y hacer clic en el botón “Save Changes”	Los cambios del perfil deberían haber sido guardados
4	Hacer clic en el botón “Upload Image” ubicado en el panel “Profile Picture” y seleccionar una imagen	La imagen de perfil de usuario debería haber sido cambiada
5	Llenar los campos en el panel “Security Information” y hacer clic en el botón	La contraseña del usuario debería haber sido cambiada

	“Change Password”	
6	Cerrar sesión e iniciar sesión con las nuevas credenciales establecidas en el paso 5	Ser redirigido a la pantalla inicial de la plataforma Beehive

Tabla 40: Casos de prueba del sprint 0

El incremento realizado en este sprint consta inicialmente de la identificación de tecnologías a utilizar, el control de sesiones y el manejo de perfil de un usuario.

La Tabla 41 muestra las tecnologías identificadas para la construcción de la plataforma Beehive.

Tecnologías Backend
• Framework Laravel para el desarrollo web y creación de la API RESTful • NodeJS con la librería SockJS para el uso de Websockets en el lado del servidor • Mosquitto Broker para la comunicación con el hardware a través del protocolo MQTT • MySql para almacenar el modelo de datos • MongoDB para almacenar los datos generados por los dispositivos
Tecnologías Frontend
• Compass para el manejo de estilos • Twitter Bootstrap para el diseño responsivo • Facebook ReactJS para la creación de componentes reutilizables en el navegador • Grunt para la unión y compresión de scripts y hojas de estilo

Tabla 41: Tecnologías identificadas para el desarrollo

La Figura 17 y la Figura 18 muestran las pantallas para el inicio de sesión y el manejo de perfil del usuario generadas en el incremento de este sprint.

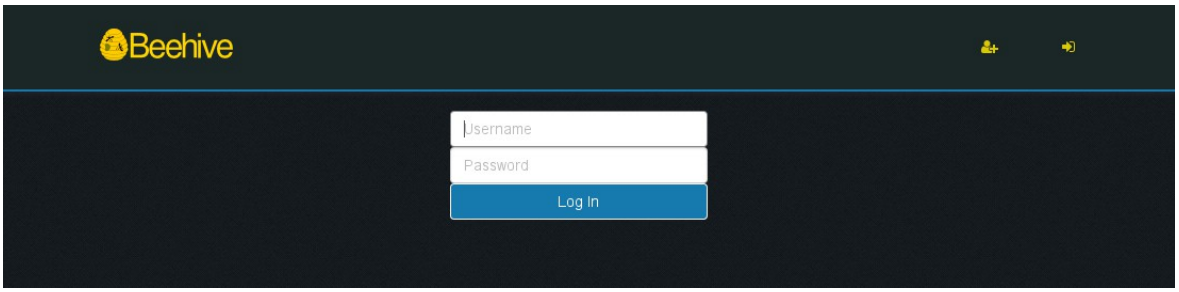

Figura 17: Inicio de sesión

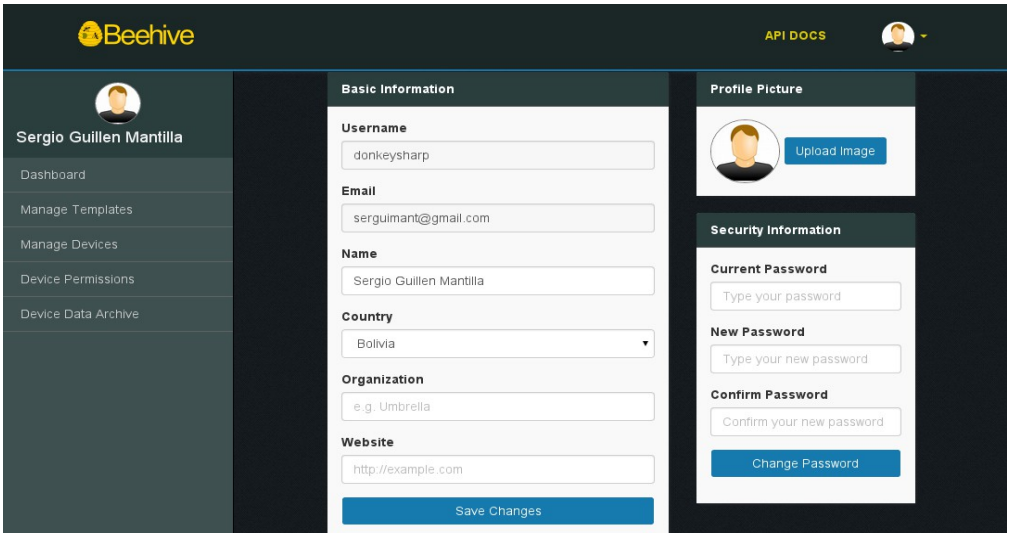

Figura 18: Edición de perfil

3.3.2. Sprint 1

En este sprint se realiza la administración de dos elementos clave del sistema, los dispositivos y los templates. Un dispositivo hereda las propiedades que cuenta un template y las propiedades que un template brinda son comandos/argumentos y data streams.

Fecha Inicio: 03 NOV 2014

Fecha Finalización: 28 NOV 2014

Sprint Backlog

La Tabla 42 muestra el backlog del Sprint 1 junto con las tareas por realizar en cada historia de usuario.

ID	Nombre de Historia de Usuario
#3	Administración de Templates
Tareas • Crear el panel de administración de Templates con el patrón Single Page utilizando ReactJS • Adicionar la lógica de negocio en los controladores • Crear servicios para Templates que retornen JSON desde el lado del servidor • Consumir los servicios desde ReactJS	
#6	Administración de Dispositivos
Tareas • Crear el panel de administración de Dispositivos con el patrón Single Page utilizando ReactJS • Crear servicios para Dispositivos que retornen JSON desde el lado del servidor • Adicionar la lógica de negocio en los controladores para los Dispositivos • Consumir los servicios de Dispositivos desde ReactJS	
#4	Administración de Comandos y Argumentos
Tareas • Dentro del panel de administración de Templates adicionar la opción de crear comandos • Dentro de la pantalla de creación de comandos, crear los controles para adicionar argumentos par aun comando • Crear servicios para Comandos y Argumentos que retornen JSON desde el lado del servidor • Adicionar la lógica de negocio en los controladores de Comandos y Argumentos usando	

transacciones para evitar datos erroneos en caso que la creación o edición falle • Consumir los servicios de Comandos y Argumentos desde ReactJS dentro del panel de Templates

Tabla 42: Backlog Sprint 1

La Tabla 43 muestra los casos de prueba realizados en este sprint.

Detalle		
Debería ser posible crear y editar el ciclo completo de un Template y Comandos/Argumentos		
Precondiciones		
• Haber iniciado sesión con un usuario válido		
#	**Pasos a realizar**	**Resultados Esperados**
1	Hacer clic en el link “Manage Templates” ubicado en el menú lateral	Ingresar a la sección de manejo de Templates que lista los templates existentes
2	Hacer clic en el botón “Add new Template”	Ingresar a la pantalla de creación de un Template
3	Llenar los datos requeridos para un template y hacer clic en el botón “Save Chages”	El template debería haber sido creado y ser redirigido a la lista de templates existentes donde el nuevo template debería aparecer
4	En el template creado hacer clic en el icono de engranajes al medio y seleccionar la opción “Commands”	Se debería expandir una sección en la que se visualicen los comandos con los que cuenta el template
5	De existir comandos editar un comando de no existir hacer clic en el botón “Add new Command”	Ir a la pantalla de edición/creación de comandos
6	Llenar los datos necesarios para comandos y argumentos y hacer clic en el botón “Save Changes”	Los datos deberían ser guardados y ser visualizados en la lista de templates
Detalle		
Debería ser posible crear y editar el ciclo completo de un Dispositivo		
Precondiciones		
• Haber iniciado sesión con un usuario válido		

• Tener al menos un Template creado		
#	**Pasos a realizar**	**Resultados Esperados**
1	Hacer clic en el link “Manage Devices” ubicado en el menú lateral	Ser redirigido a la pantalla donde se listan los dispositivos filtrados por template
2	Hacer clic en el botón “Add new Device”	Ser redirigido a la pantalla de creación de un dispositivo
3	Llenar los campos requeridos además de seleccionar el Template al cual pertenecerá el dispositivo a ser creado y hacer clic en el botón “Save Changes”	Un nuevo dispositivo debería haber sido creado y ser visible en la lista de dispositivos habiendo filtrado el template al cual pertenece el dispositivo

Tabla 43: Casos de prueba del sprint 1

La Figura 19, Error: no se encontró el origen de la referencia, Figura 21, Figura 22 y la Figura 23 muestran las pantallas para la administración de templates con sus respectivos comandos y la administración de dispositivos.

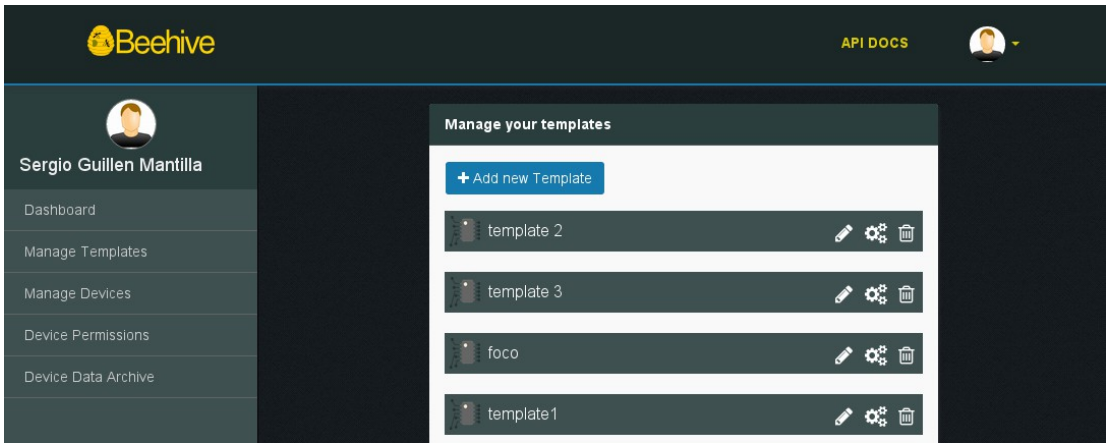

Figura 19: Listado de templates

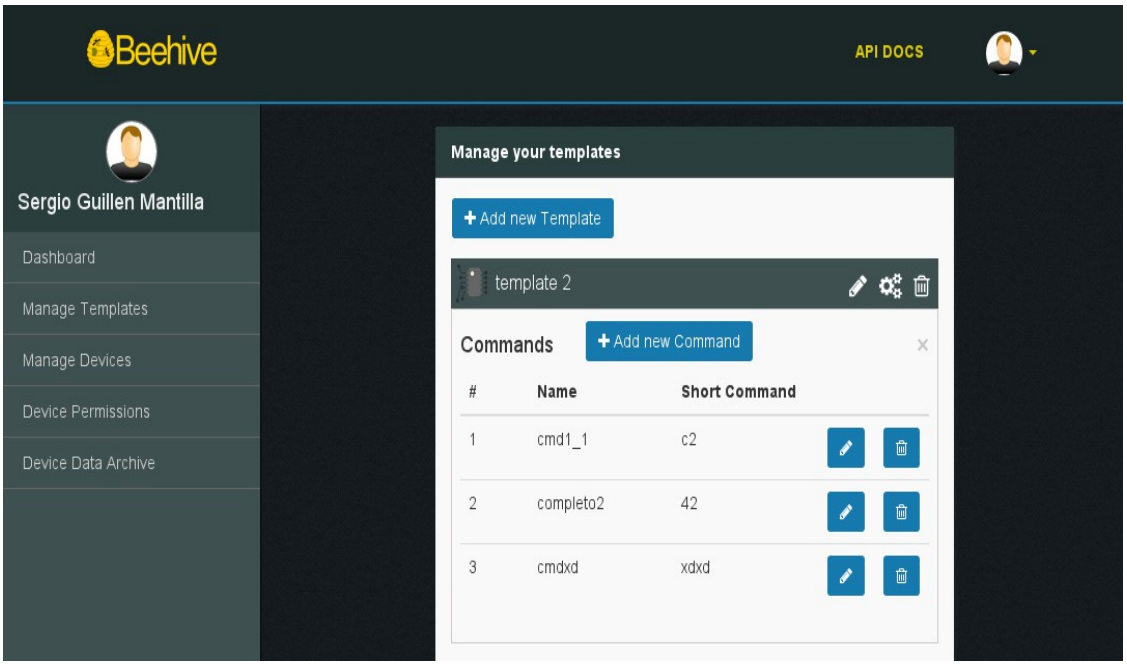

Figura 20: Comandos de un template

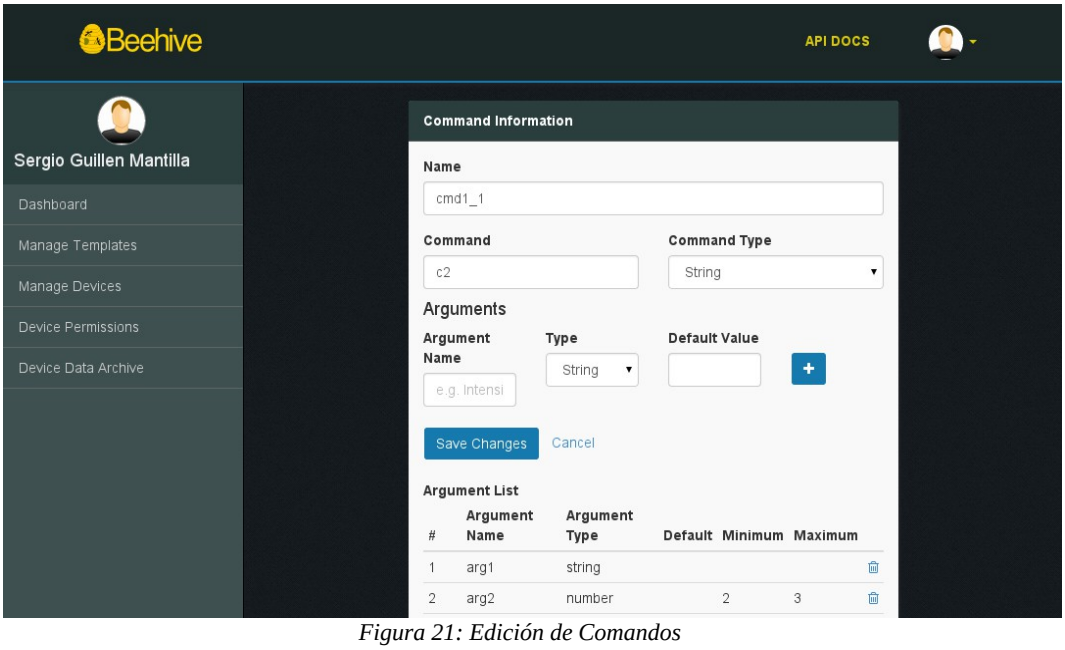

Figura 21: Edición de Comandos

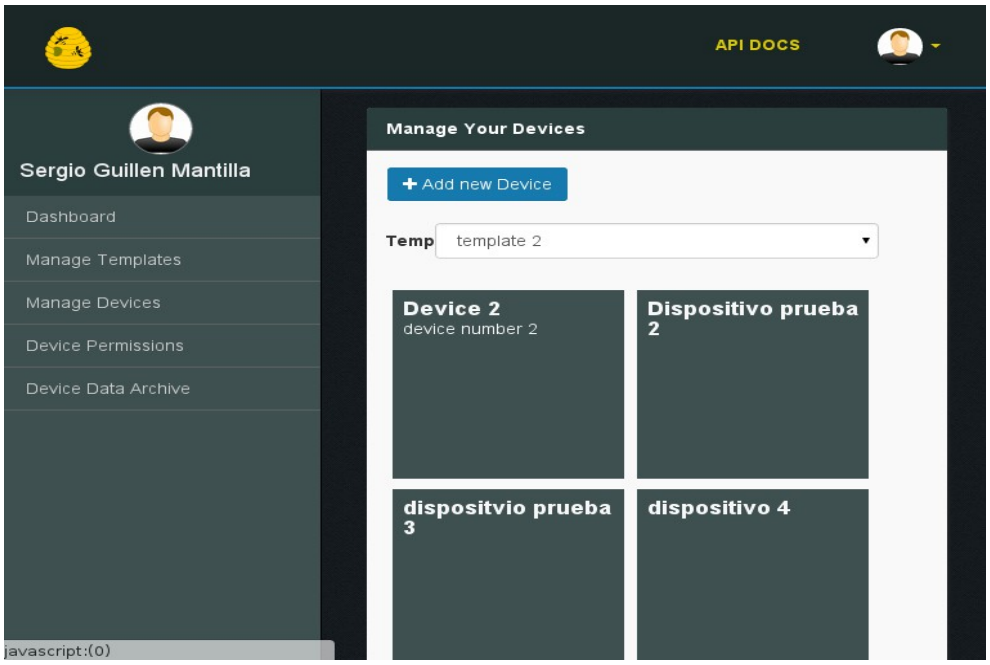

Figura 22: Listado de dispositivos filtrados por template

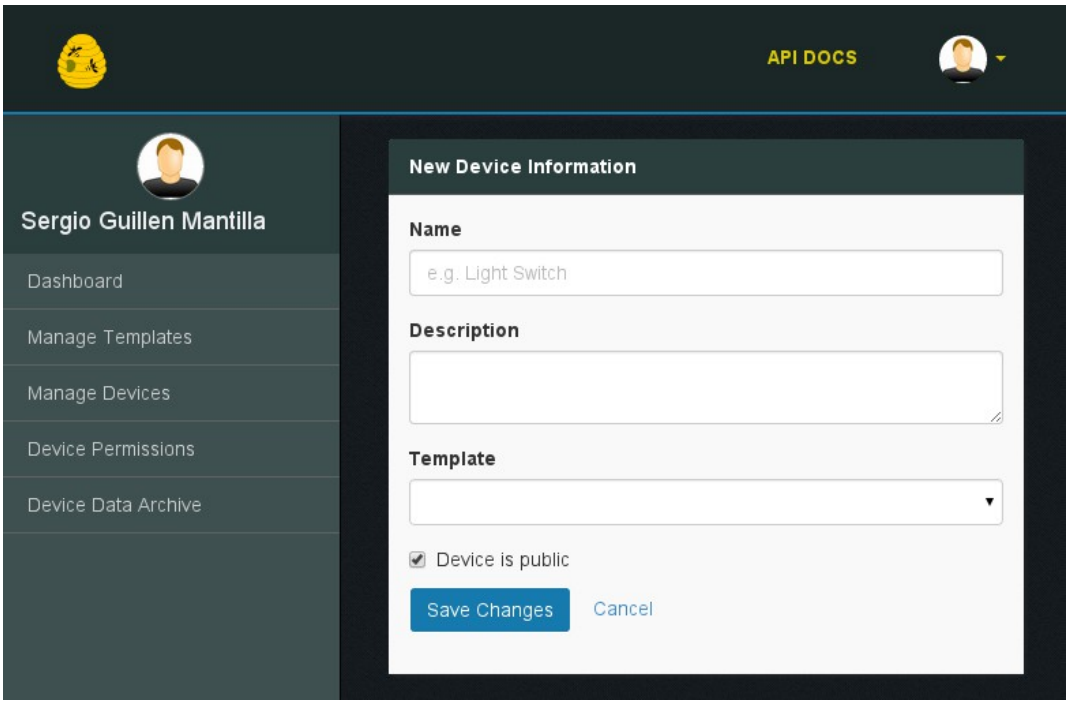

Figura 23: Creación de dispositivo

3.3.3. Sprint 2

Este sprint brinda funcionalidad a los canales de comunicación, además que estos logren integrarse con la plataforma web de tal forma que los datos enviados por dispositivos de hardware pueden visualizarse en la plataforma web en tiempo real.

Fecha Inicio: 01 DEC 2014

Fecha Finalización: 19 DEC 2014

Sprint Backlog

La Tabla 44 muestra el backlog del Sprint 2 junto con las tareas por realizar en cada historia de usuario.

ID	Nombre de Historia de Usuario
#9	Implementación de los Canales de Comunicación
Tareas • Montar el broker MQTT Mosquitto • Configurar broker Mosquitto • Documentar la instalación y configuración de Mosquitto	
#33	Seguridad en los Canales de Comunicación
Tareas • Compilar, instalar y configurar el plugin "mosquitto-auth-plug" • Crear los servicios en la plataforma web para que "mosquitto-auth-plug" pueda comunicarse con esta	
#37	Integración entre los Canales de Comunicación y la Plataforma Web
Tareas • Montar un servidor de websockets con NodeJS • Consumir los canales de comunicación MQTT desde NodeJS • Crear una librería javascript en la plataforma web para comunicarse por websockets	

Tabla 44: Backlog Sprint 2

La Tabla 45 muestra los casos de prueba realizados a los canales de comunicación que son parte del incremento en este sprint.

Detalle		
Debería ser posible iniciar y terminar sesión en la plataforma Beehive		
Precondiciones		
Ninguna		
#	**Pasos a realizar**	**Resultados Esperados**
1	Hacer uso de la herramienta mosquitto_sub para poder suscribirse a un tópico MQTT con usuario y password válidos	La herramienta debería poder ejecutarse con normalidad y estar a la espera de nuevos mensajes
2	Hacer uso de la herramienta mosquitto_pub para poder enviar mensajes a mismo tópico en el que se realizó la suscripción en el paso 1	Los mensajes enviados con la herramienta mosquitto_pub deberían ser visualizados por la herramienta ejecutándose en el paso 1

Tabla 45: Casos de prueba del sprint 2

Este sprint no cuenta con incrementos visuales pero sí funcionales que permiten a los dispositivos de hardware la comunicación con la plataforma Beehive.

3.3.4. Sprint 3 – Refactorización

Al ser necesario el mantenimiento, reusabilidad y extensibilidad del código es necesario reestructurar la plataforma Beehive usando principios S.O.L.I.D. Es por ello que este sprint realiza el proceso de refactorización del código de la plataforma web haciendo uso de los principios S.O.L.I.D.

Una nueva historia de usuario se adiciona al backlog de historias de usuarios. La Tabla 46 describe a la historia de usuario que permite la refactorización de la plataforma web.

	Historia de Usuario
Número: #47	**Nombre:** Refactorizar plataforma web usando principios S.O.L.I.D.
Puntos: 8	**Usuario:** Desarrollador
Descripción: Refactorizar plataforma web usando principios S.O.L.I.D.	

Tabla 46: H.U. Refactorizar plataforma web usando principios S.O.L.I.D.

Fecha Inicio: 29 DEC 2014

Fecha Finalización: 14 FEB 2015

Sprint Backlog

La Tabla 47 muestra el backlog del Sprint 3 junto con las tareas por realizar en cada historia de usuario.

ID	Nombre de Historia de Usuario
#47	Refactorizar plataforma web usando principios S.O.L.I.D.
Tareas • Refactorizar toda la estructura de la plataforma web haciendo uso del patrón "Repository Pattern" • Para permitir la extensibilidad, reusabilidad y mantenimiento aplicar principios S.O.L.I.D.	

Tabla 47: Backlog Sprint 3

Al ser este un sprint de refactorización no existe un incremento visual, mas sí un incremento a nivel de estructura de código que permitirá la reusabilidad y mantenibilidad de este.

Los casos de prueba que se ejecutan en este sprint son los mismos planteados anteriormente.

3.3.5. Sprint 4

En este sprint se cuenta con un panel en el que se puedan realizar las siguientes tareas sobre un dispositivo:

- Usar las librerías creadas en el Sprint 2 (ver sección 3.4.3) que integran los canales de comunicación con la plataforma web para poder así visualizar los datos de un dispositivo de hardware en tiempo real con diferentes tipos de visualización (gráficas de barras, líneas, mapas, imágenes y texto estático)
- Enviar comandos a un dispositivo desde un panel de ejecución de comandos

Fecha Inicio: 23 FEB 2015

Fecha Finalización: 31 MARZO 2015

Sprint Backlog

La Tabla 48 muestra el backlog del Sprint 4 junto con las tareas por realizar en cada historia de usuario.

ID	Nombre de Historia de Usuario
#5	Administración de los Data Streams para un Template
Tareas • Adicionar pantalla para crear data streams desde el panel de Templates • Administrar la creación, modificación y eliminación de data streams • Consumir servicios JSON desde javascript	
#8	Panel de visualización de Data Streams de un Dispositivo
Tareas • Crear servicios en el servidor que devuelvan JSON con información de los data streams • Usar React JS para crear componentes re utilizables que muestren los data streams en gráficas de tipo línea • Usar React JS para crear componentes re utilizables que muestren los data streams en gráficas de tipo barra • Usar React JS para crear componentes reutilizables que muestren los data streams en gráficas de tipo mapa • Usar React JS para crear componentes reutilizables que muestren los data streams en gráficas de tipo imágenes codificadas en base64 • Consumir servicios JSON de los data streams de datos • Mostrar los data streams de datos a través de websockets	
#7	Panel de Comandos par aun Dispositivo
Tareas • Obtener desde del servidor los comandos y argumentos asociados a un Dispositivo • Crear con React JS componentes re utilizables para la ejecución de comandos con sus argumentos si es que existen • Crear del lado del servidor un mecanismo para conectar la información del comando ejecutado con los Canales de Comunicación	

Tabla 48: Backlog Sprint 4

La Tabla 49 muestra los casos de pruebas realizados en este sprint.

Detalle		
Debería ser posible crear y editar el ciclo completo de un Template y sus Data Streams		
Precondiciones		
• Haber iniciado sesión con un usuario válido • Tener un template creado		
#	**Pasos a realizar**	**Resultados Esperados**
1	Hacer clic en el link “Manage Templates” ubicado en el menú lateral	Ingresar a la sección de manejo de Templates que lista los templates existentes
2	En algún template listado hacer clic en el icono de engranajes al medio y seleccionar la opción “Data Streams”	Se debería expandir una sección en la que se visualicen los data streams con los que cuenta el template
3	Hacer clic en el botón “Add new Data Stream”	Ir a la pantalla de edición/creación de data streams
4	Llenar los campos requeridos para el data stream, es importante notar el nombre del tópico ya que este será utilizado en los canales de comunicación. Además es importante seleccionar el tipo de dato que recibirá este data stream. Hacer clic en el botón “Save Changes”	Un nuevo data stream debería haber sido creado
Detalle		
Debería ser posible visualizar data streams en tiempo real a través del panel de detalles de un dispositivo		
Precondiciones		
• Haber iniciado sesión con usuario válido • Tener un dispositivo a un template el cual tenga al menos un data stream		
#	**Pasos a realizar**	**Resultados Esperados**
1	Hacer clic en el link “Manage Devices” ubicado en el menú lateral	Visualizar una lista de dispositivos filtrados por template

2	Hacer clic en el dispositivo seleccionado	El panel “Device Data Streams” debería mostrar los data streams definidos en el template al cual pertenece el dispositivo
3	Del panel “Device Information” seleccionar y copiar el campo “MQTT-PUB Key”	
4	Del template al cual pertenece el dispositivo seleccionar y copiar el campo de “Topic Name” de algún data stream definido en este	
5	Generar una cadena concatenando de la siguiente forma {MQTT-PUB Key}/{Topic Name}	
6	Hacer uso de la herramienta mosquitto_pub y enviar datos válidos al tópico obtenido en el paso 5	Los datos enviados con la herramienta mosquitto_pub deberían ser visualizados en el panel gráfico correspodiente al data stream en el panel “Device Data Streams”
Detalle		
Debería ser posible ejecutar comandos a través del panel de detalles de un dispositivo		
Precondiciones		
• Haber iniciado sesión con usuario válido • Tener un dispositivo a un template el cual tenga al menos un comando		
#	**Pasos a realizar**	**Resultados Esperados**
1	Hacer clic en el link “Manage Devices” ubicado en el menú lateral	Visualizar una lista de dispositivos filtrados por template
2	Hacer clic en el dispositivo seleccionado	El panel “Device Data Streams” debería mostrar los data streams definidos en el template al cual pertenece el dispositivo
3	Del panel “Device Information” seleccionar y copiar el campo “MQTT-SUB Key”	
4	Generar una cadena concatenando de la siguiente forma {MQTT-PUB Key}/**command**	

5	Hacer uso de la herramienta mosquitto_sub para visualizar los mensajes enviados al tópico generado en el paso 4	
6	Ejecutar un comando que se encuentra en el panel “Device Commands” que se encuentra. Puede ser un comando con o sin argumentos	La información del comando ejecutado debería ser visualizada por la aplicación mosquitto_pub ejecutada en el paso 5

Tabla 49: Casos de prueba del sprint 4

La Figura 24, Figura 25, Figura 27 y la Figura 26 muestran el panel de administración de un dispositivo que muestra la visualización en tiempo real de data streams y la ejecución de comandos.

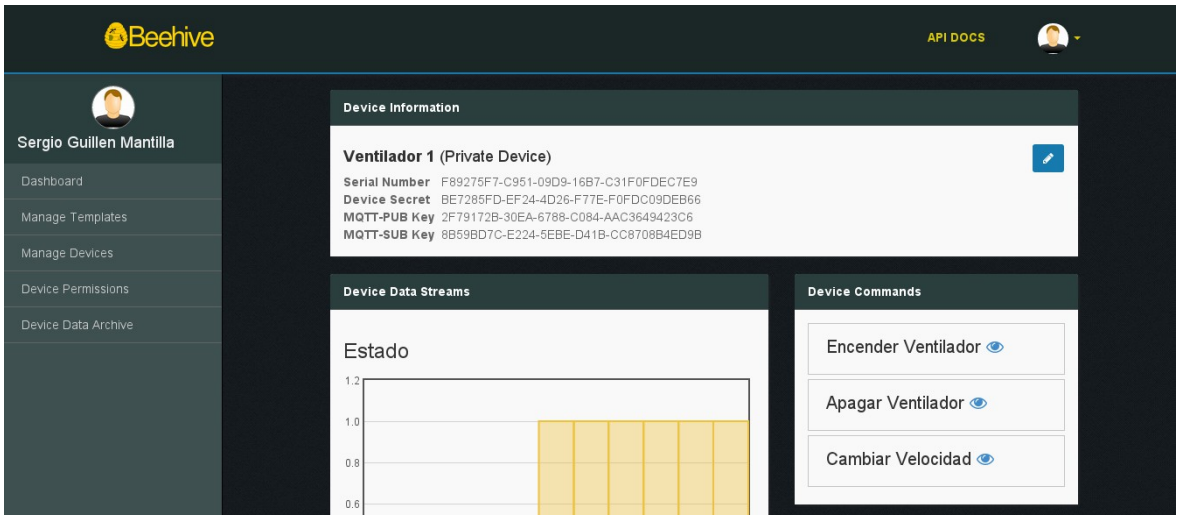

Figura 24: Panel de administración de un dispositivo

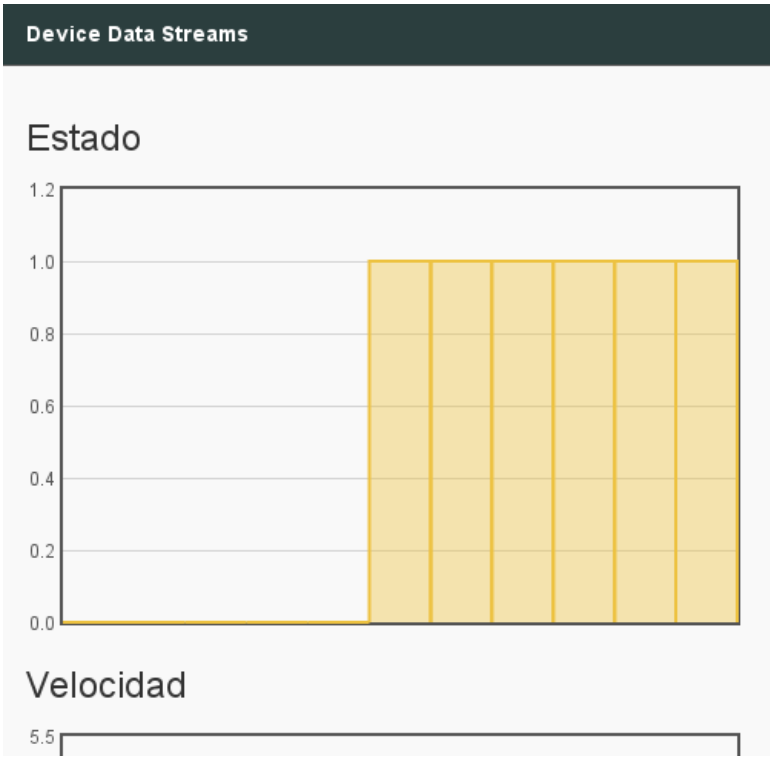

Figura 25: Data Stream de tipo barra

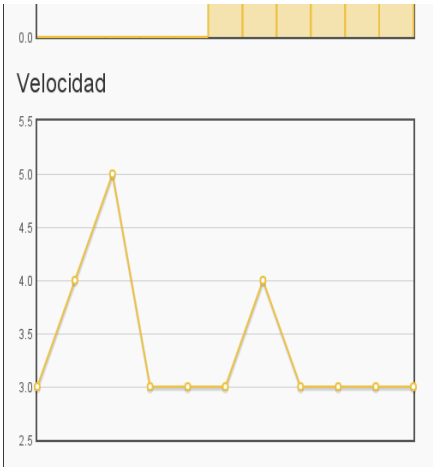

Figura 27: Data Stream de tipo línea

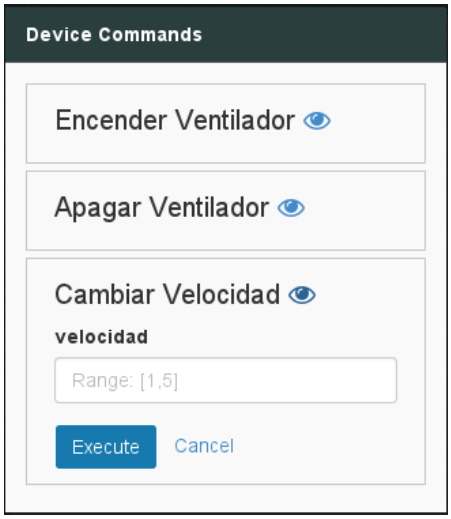

Figura 26: Comandos de un dispositivo

3.3.6. Sprint 5

En este sprint se realiza el desarrollo de los servicios RESTful para la plataforma Beehive para administrar dispositivos, templates, comandos/argumentos y data streams, de tal forma que estos datos puedan ser consumidos por otros desarrolladores desde otras plataformas.

Fecha Inicio: 31 MAR 2015

Fecha Finalización: 14 ABR 2015

Sprint Backlog

La Tabla 50 muestra el backlog del Sprint 5 junto con las tareas por realizar en cada historia de usuario.

ID	Nombre de Historia de Usuario
#11	Servicios RESTful para Autenticación
Tareas • Utilizar Json Web Tokens para permitir la autenticación en los servicios RESTful • Crear los endpoints para la autenticación • Crear los filtros para verificar si un token es válido	
#58	Servicios RESTful para Dispositivos
Tareas • Crear endpoint para la lectura de dispositivos • Crear endpoint para la lectura de un dispositivo específico • Crear endpoint para crear dispositivos • Crear endpoint para actualizar dispositivo	
#59	Servicios RESTful para templates
Tareas • Crear endpoint para la lectura de templates • Crear endpoint para la lectura de un template específico • Crear endpoint para crear templates • Crear endpoint para modificar un template	
#60	Servicios RESTful para comandos y argumentos
Tareas	

• Crear dos endpoints para leer comandos y argumentos • Crear dos endpoints para leer comandos y argumentos específicos • Crear dos endpoints para crear comandos y argumentos • Crear endpoint para modificar comandos • Crear dos endpoints para eliminar comandos y argumentos	
#61	Servicios RESTful para data streams
Tareas • Crear endpoint para leer varios data streams • Crear endpoint para leer un data stream específico • Crear endpoint para crear data streams • Crear endpoint para modificar un data stream • Crear endpoint para eliminar un data stream	

Tabla 50: Backlog Sprint 5

La Tabla 51 muestra los casos de prueba de este sprint.

Detalle		
Debería ser posible iniciar y terminar sesión en la plataforma Beehive a través de la API RESTful		
Precondiciones		
Ninguna		
#	**Pasos a realizar**	**Resultados Esperados**
1	Haciendo uso de la herramienta Advanced REST Client enviar una petición POST al endpoint de autenticación con los campos de username y password con datos válidos	Se debería obtener la respuesta en formato JSON con un atributo llamado token que será utilizado para futuras peticiones a la API RESTful
Detalle		
Debería ser posible cumplir el ciclo de los recursos templates, comandos, data streams y dispositivos a través de la API RESTful		
Precondiciones		
• Tener un token válido		
#	**Pasos a realizar**	**Resultados Esperados**
1	Haciendo uso de la herramienta Advanced	Obtener una lista de recursos en formato

	REST Client enviar una petición GET al endpoint del recurso elegido.	JSON
2	Haciendo uso de la herramienta Advanced REST Client enviar una petición GET al endpoint del recurso elegido proporcionando el identificador de un recurso existente.	Obtener un recurso específico en formato JSON
3	Haciendo uso de la herramienta Advanced REST Client enviar una petición POST al endpoint del recurso elegido.	Debería crear un recurso
4	Haciendo uso de la herramienta Advanced REST Client enviar una petición PUT al endpoint del recurso elegido con el identificador de un recurso existente.	Debería actualizar un recurso específico

Tabla 51: Casos de prueba del sprint 5

El incremento generado en el presente sprint es la API RESTful la cual no tiene características visibles pero sí funcionales y utilizables por otros desarrolladores. El Anexo B muestra la documentación de la API RESTful que puede ser utilizada por otros desarrolladores con el fin de extender la plataforma Beehive a otras plataformas.

3.3.7. Sprint 6

En este sprint se desarrolla la lógica y vistas que permiten a un usuario administrar permisos a otros usuarios sobre dispositivos, de este modo es posible compartir dispositivos de hardware entre usuarios. El presente sprint generan dos historias de usuario para la administración de permisos sobre dispositivos y visualización de dispositivos compartidos. La Tabla 52 y la Tabla 53 muestran las nuevas historias de usuario.

Fecha Inicio: 20 ABR 2015

Fecha Finalización: 24 ABR 2015

	Historia de Usuario
Número: #85	**Nombre:** Administrar permisos sobre dispositivos
Puntos: X	**Usuario:** Usuario Web
Descripción: Como usuario deseo poder otorgar permisos (lectura, escritura y ejecución) sobre los dispositivos que poseo de tal forma que pueda compartir el acceso a estos dispositivos	

Tabla 52: Administrar permisos sobre dispositivos

	Historia de Usuario
Número: #86	**Nombre:** Visualizar dispositivos compartidos
Puntos: 5	**Usuario:** Usuario Web
Descripción: Como usuario deseo ver los dispositivos compartidos de tal forma que pueda acceder al panel de visualización de datos en tiempo real como editar los permisos de este.	

Tabla 53: Visualizar dispositivos compartidos

Sprint Backlog

La Tabla 54 muestra el backlog del sprint 6.

ID	Nombre de Historia de Usuario
#85	Administrar permisos sobre dispositivos
Tareas • Crear las pantallas para visualizar los dispositivos filtrados por template y editar los permisos de este • Crear los métodos en el servidor para cambiar los servicios de este verificando si el usuario puede cambiarlo	
#86	Visualizar dispositivos compartidos
Tareas • Crear un nuevo filtro de dispositivos compartidos en las pantallas de dispositivos y permisos sobre dispositivos • Crear un método en el servidor que retorne los dispositivos compartidos de un usuario autenticado	

Tabla 54: Backlog Sprint 6

La Tabla 55 muestra los casos de prueba utilizados en el sprint 6.

Detalle		
Debería ser posible adicionar permisos sobre un dispositivo		
Precondiciones		
• Tener al menos dos usuarios • Un usuario debería tener al menos un dispositivo que cuente con al menos un data stream y un comando		
#	**Pasos a realizar**	**Resultados Esperados**
1	Hacer clic en el link “Device Permissions” ubicado en el menú lateral	Debería desplegar una lista de dispositivos filtrados por template
2	Hacer clic en el ícono en forma de candado de un dispositivo para administrar los permisos de este	Debería ingresar a la pantalla de edición de permisos donde al menos el usuario dueño del dispositivo debería ser mostrado en la tabla
3	Escribir el username del usuario al que se desea otorgar permisos y combinar los diferentes permisos. Hacer clic en el botón de adicionar	El usuario adicionado debería aparecer en la tabla de administradores.
4	Ingresar con el usuario al que se otorgaron permisos y hacer clic en el link “Manage Devices” en el menú lateral	En el filtro de template debería haber una opción de dispositivos compartidos
5	Seleccionar la opción de dispositivos compartidos	Debería aparecer el dispositivo que se compartió en el paso 3
6	Ingresar al dispositivo compartido para editar los datos del dispositivo y ejecutar comandos.	En base a los permisos que se otorgaron en el paso 3 se debería o no editar o ejecutar.

Tabla 55: Casos de prueba del sprint 6

3.4. Modelo de Datos

Hacer uso de una metodología de desarrollo de software basada en Scrum (Open Scrum, ver sección 2.6.3.) permite adaptarse a diferentes cambios que puedan generarse (a nivel de lógica

de negocio, modelo de datos, arquitectura, diseño y otros). La Figura 28 muestra el diagrama relacional que representa el modelo de datos que usa la plataforma Beehive tras haber concluido siete iteraciones (sprints).

3.5. Diagrama de clases

Antes del sprint 3 en el que se hace una refactorización del código para el núcleo de la plataforma Beehive, el diagrama de clases solo contaba con las clases que representan los conceptos mencionados al inicio del capítulo. Tras haber realizado la refactorización el uso de principios de diseño S.O.L.I.D. permiten abstraer la lógica de negocio en diferentes clases independientemente de la plataforma web. Los principios S.O.L.I.D. recomiendan que el modelo se realice en base a abstracciones (interfaces) para que el modelo no dependa directamente de clases.

La Figura 29 muestra el diagrama de clases del núcleo de la plataforma basado en abstracciones.

3.6. Diagrama de Despliegue

Concluidos los siete sprints de desarrollo se puede dividir la arquitectura de la plataforma Beehive en tres nodos físicos que son independientes uno del otro y solo dependencias de las abstracciones. La Figura 30 muestra el diagrama de despliegue de la plataforma Beehive.

3.7. Distribución de Software

Principalmente un proyecto Open-Source debe tener publicado el código fuente, además de ser distribuido bajo una licencia Open-Source compatible.

3.7.1. Distribución de Código

El código fuente y la documentación del proyecto Beehive se encuentra publicado en el repositorio de proyectos GitHub https://github.com/BeehiveIOT. Dentro de este proyecto se encuentran dos sub-proyectos.

- Beehive-web – Contiene la plataforma web como tal y el núcleo de la aplicación

https://github.com/BeehiveIOT/beehive-web

- Beehive-bridge – Se encarga de enlazar los canales de comunicación con el núcleo de la plataforma (beehive-web) https://github.com/BeehiveIOT/beehive-bridge

3.7.2. Licencia de Distribución

La licencia Open-Source con la que el proyecto Beehive será distribuido es la licencia MIT. Esta licencia es parte de la categoría de licencias Open-Source permisivas la cual entre sus principales características permite al usuario o compañía que desee usar este software: Uso, distribución, modificación, publicación, sub licenciamiento y/o venta de copias del software.

3.8. Implementación de una instancia en la nube para pruebas

Para poder realizar las pruebas (ver capítulo 4) la plataforma Beehive debe encontrarse en un servidor público o una instancia en la nube.

La instancia en la que se realizaran las pruebas para la plataforma Beehive se encuentra en la siguiente ubicación: http://iot.tiendamerx.com

Los datos de prueba para un usuario de la plataforma web son:

- Usuario: *user1, user2, user3, user4 y user 5*
- Password: *12345*

La Tabla 56 muestra las características del servidor en la nube.

Característica	
RAM	2GB
Núcleos	2
Disco SSD	40GB
Sistema Operativo	GNU/Linux Debian
Servidor HTTP	Nginx

Tabla 56: Características del servidor en la nube

La Tabla 57 muestra as aplicaciones instaladas en esta instancia en la nube para poder ejecutar la plataforma Beehive.

Aplicación	**Descripción**
Nginx	Servidor HTTP y balanceador de carga para alojar la plataforma web y la API RESTful
Mosquitto	Broker MQTT para los canales de comunicación
Mosquitto-auth-plugin	Complemento para Mosquitto para la autorización de dispositivos
NodeJS y Forever	Necesarios para ejecutar el mecanismo de comunicación en tiempo real con la plataforma web y los canales de comunicación

Tabla 57: Aplicaciones instaladas en instancia en la nube

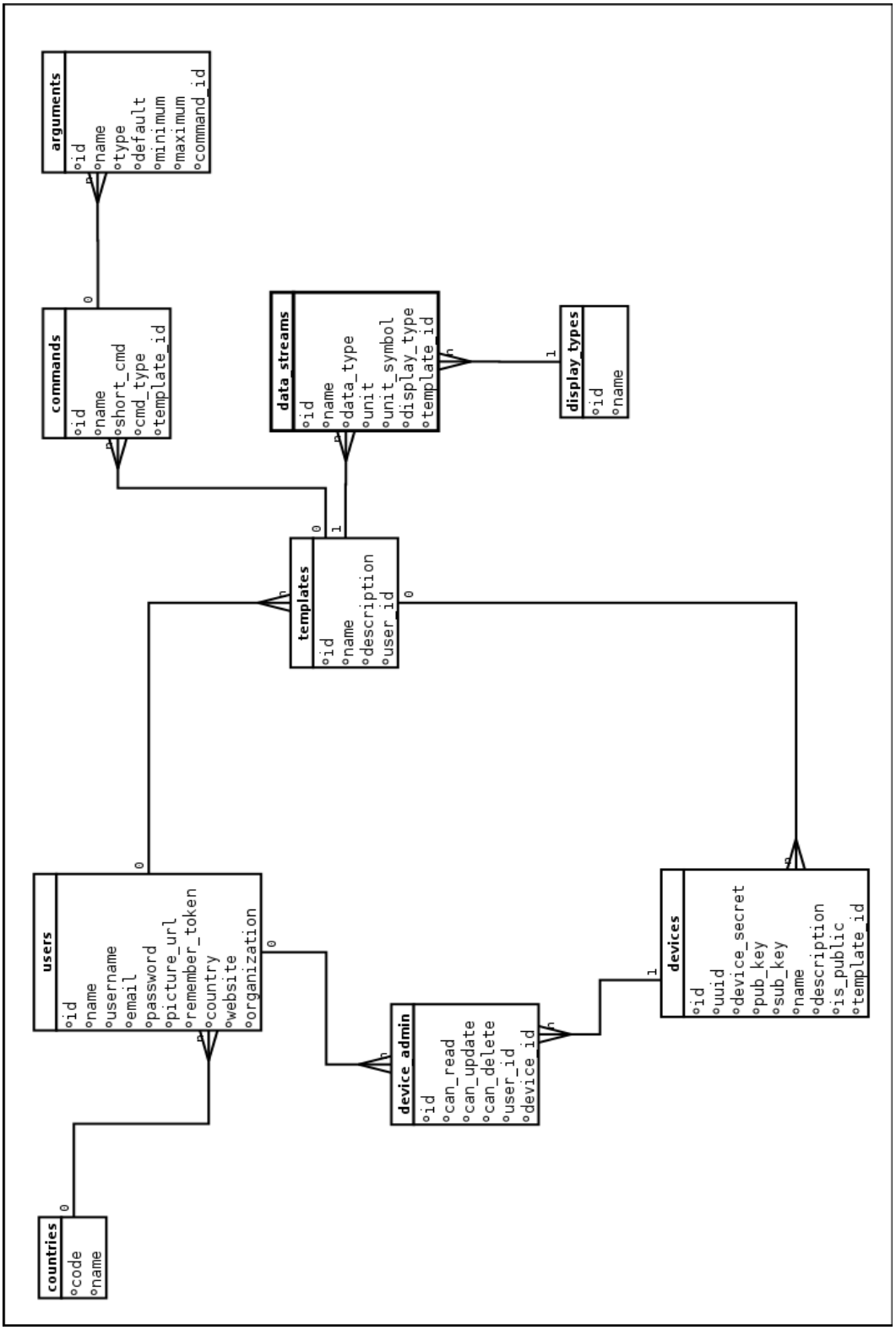

Figura 28: Modelo Relacional de la plataforma Beehive

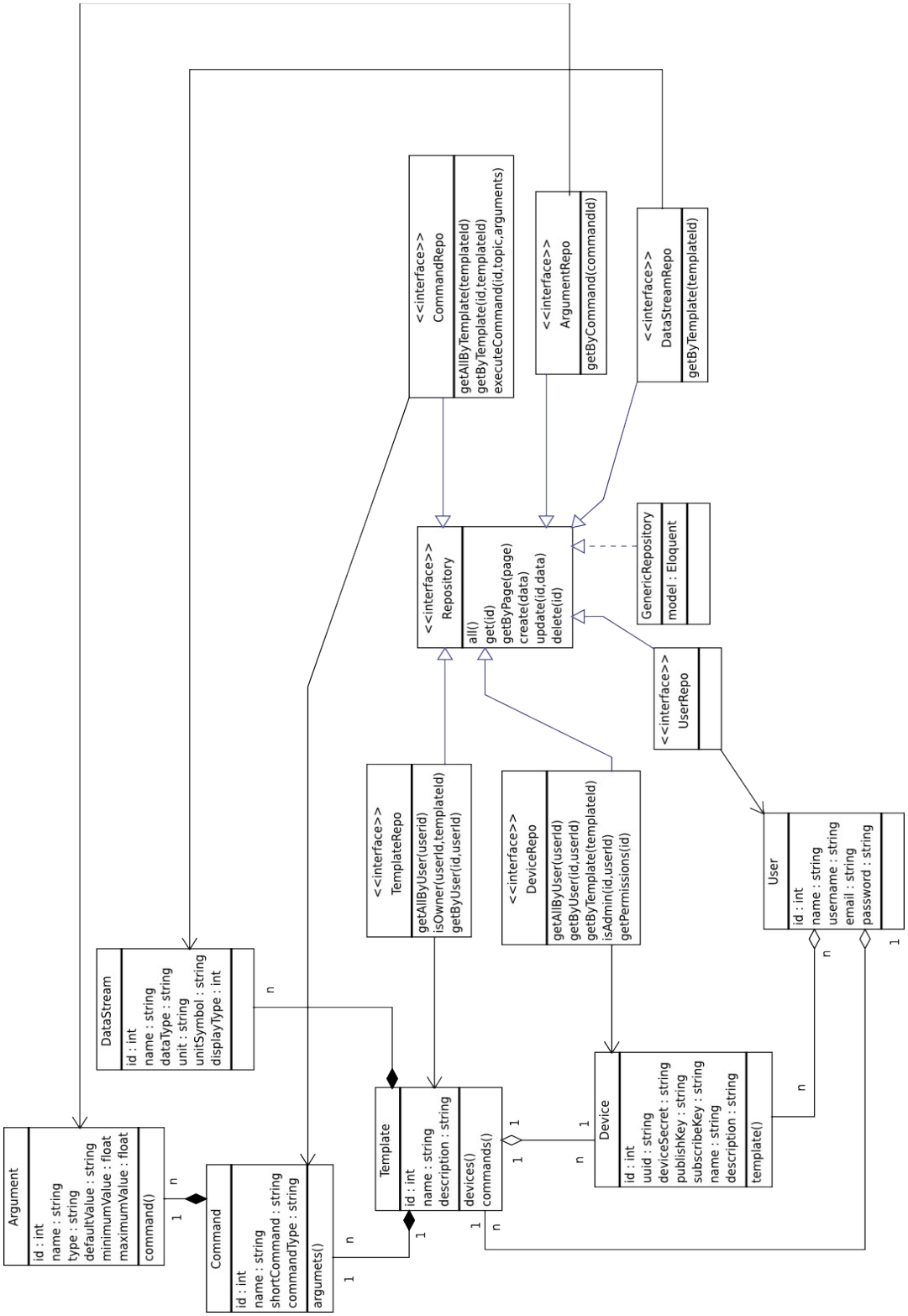

Figura 29: Diagrama de clases considerando solamente abstracciones

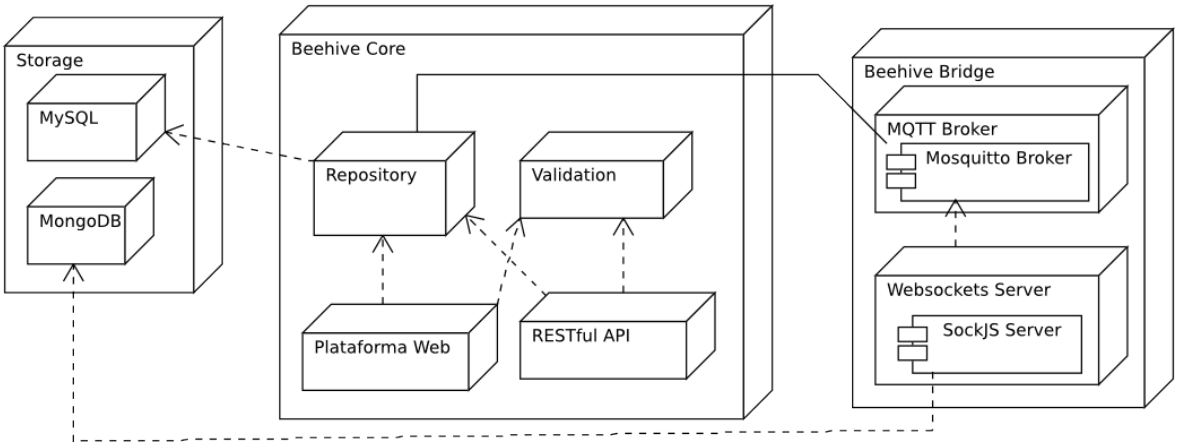

Figura 30: Diagrama de despliegue de la plataforma Beehive

CAPÍTULO 4
PRUEBAS Y RESULTADOS

Para mostrar la funcionalidad de la plataforma Beehive se realizan pruebas a nivel de hardware y pruebas de estrés a nivel de software. Las pruebas a nivel de hardware muestran como dispositivos de hardware (Arduino y Raspberry Pi) interactúan con la plataforma mediante los canales de comunicación. Por otro lado las pruebas de estrés hacia el software permiten ver la reacción de la plataforma Beehive ante una determinada cantidad de peticiones concurrentes.

4.1. Pruebas con Dispositivos de Hardware

4.1.1. Prueba Domótica (Sensor/Actuador)

Esta prueba muestra como un conjunto de sensores y actuadores distribuidos en un supuesto departamento (maqueta) interactúan en tiempo real con la plataforma Beehive.

Descripción

La Figura 35 muestra la distribución de los sensores y actuadores en la maqueta de un departamento. El sensor de gas y humo se encuentra en la cocina la cual es más proclive a generar escapes de gas o exceso de humo. La Figura 31 muestra como se visualizan los datos en tiempo real dentro de la plataforma Beehive para el sensor de humo, la figura muestra un pico tras haber sido detectada la presencia de humo en el ambiente.

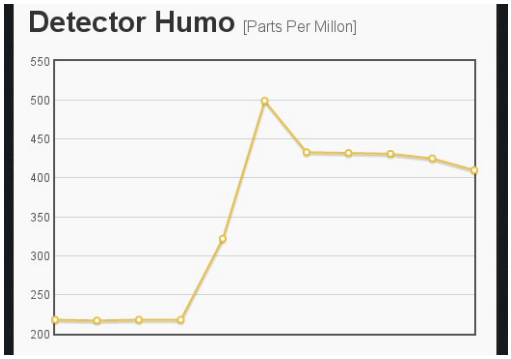

Figura 31: Gráfica del sensor de gas y humo

De la misma forma, el sensor de temperatura y humedad ubicado en la habitación muestra la

temperatura y el nivel de humedad en el ambiente. La Figura 32 muestra como se visualizan los datos en tiempo real dentro de la plataforma Beehive para este sensor, gran parte del tiempo los datos de este sensor se mantienen constante.

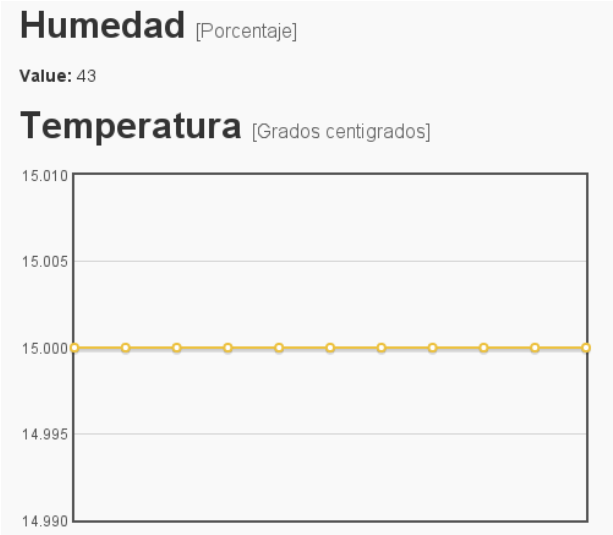

Figura 32: Gráfica del sensor de temperatura y humedad

Para detectar si se la puerta se abre un sensor ultrasónico que puede medir la distancia es utilizado, en la maqueta al estar la puerta cerrada la distancia desde la ubicación del sensor hasta la pared es mayor a los 30 cm al abrirse la puerta esta distancia se reduce y es cuando la cámara actúa y realiza el envío de imágenes codificadas en base64 a la plataforma Beehive. La Figura 33 muestra como se visualizan los datos en tiempo real dentro de la plataforma Beehive para este sensor, la figura muestra un cambio en las distancias tras haber abierto la puerta.

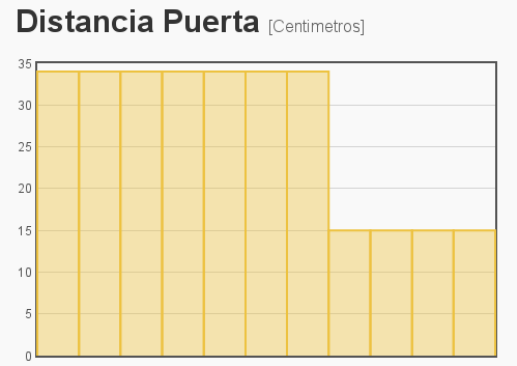

Figura 33: Gráfica para el sensor de distancia

Finalmente para el control de las lámparas (en el caso de la maqueta solo existe una) existen dos comandos, uno para encender y otro para apagar las luces.

Componentes Necesarios

La Tabla 58 muestra los componentes necesarios para esta prueba.

Dispositivo	**Descripción**
Sensor HC-SR04	Sensor digital ultrasónico que permite medir distancias (verificar si una puerta se abre)
Sensor MQ-2	Sensor análogo que detecta la presencia de fugas de gas o humo en un ambiente
Sensor DHT-11	Sensor digital que mide la humedad y temperatura del ambiente
LDR	Resistencia que genera datos análogos en base a la luz, a más luz menos resistencia y opuestamente a menos luz más resistencia
Cámara Web	Trabaja como sensor y actuador para la captura de imágenes en caso se abra la puerta
Lámpara	Trabaja como actuador que puede ser controlado remotamente por la plataforma Beehive

Tabla 58: Sensores y actuadores necesarios para la prueba

Condiciones iniciales en la plataforma Beehive

Dentro de la plataforma Beehive es necesario crear un Template con los comandos/argumentos y data streams necesarios que permitan llevar a cabo las tareas del dispositivo de prueba, la Tabla 59 muestra los detalles del Template.

NOMBRE DE TEMPLATE: Tesis Smarthome
COMANDOS
Nombre: Encender lámpara **Tipo de Comando:** String **Comando:** “1”
Nombre: Apagar lámpara

Tipo de Comando: String **Comando:** “0”
DATA STREAMS
Nombre: Humedad **Nombre de Tópico:** humidity **Tipo de dato:** Número **Tipo de gráfica:** Texto Estático **Unidad:** ninguna **Símbolo de unidad:** ninguna
Nombre: Temperatura **Nombre de Tópico:** temperature **Tipo de dato:** Número **Tipo de gráfica:** Lineas **Unidad:** Grados centígrados **Símbolo de unidad:** °C
Nombre: Detector Humo **Nombre de Tópico:** temperature **Tipo de dato:** Número **Tipo de gráfica:** Lineas **Unidad:** Partes Por Millón **Símbolo de unidad:** PPM
Nombre: Distancia Puerta **Nombre de Tópico:** distance **Tipo de dato:** Número **Tipo de gráfica:** Barras **Unidad:** Centímetros **Símbolo de unidad:** cm
Nombre: Intensidad de la luz **Nombre de Tópico:** light **Tipo de dato:** Número **Tipo de gráfica:** Texto estático

Unidad: Ohmnios **Símbolo de unidad:** Ohm

Tabla 59: Descripción de template para dispositivo de hardware (sensor/actuador)

Diseño de dispositivo de hardware

El dispositivo de hardware para esta prueba principalmente hace uso de un Arduino Mini Pro para la lectura de datos de los sensores y un Raspberry Pi para los actuadores y la comunicación con la plataforma Beehive.

La Figura 34 muestra el circuito necesario para esta prueba.

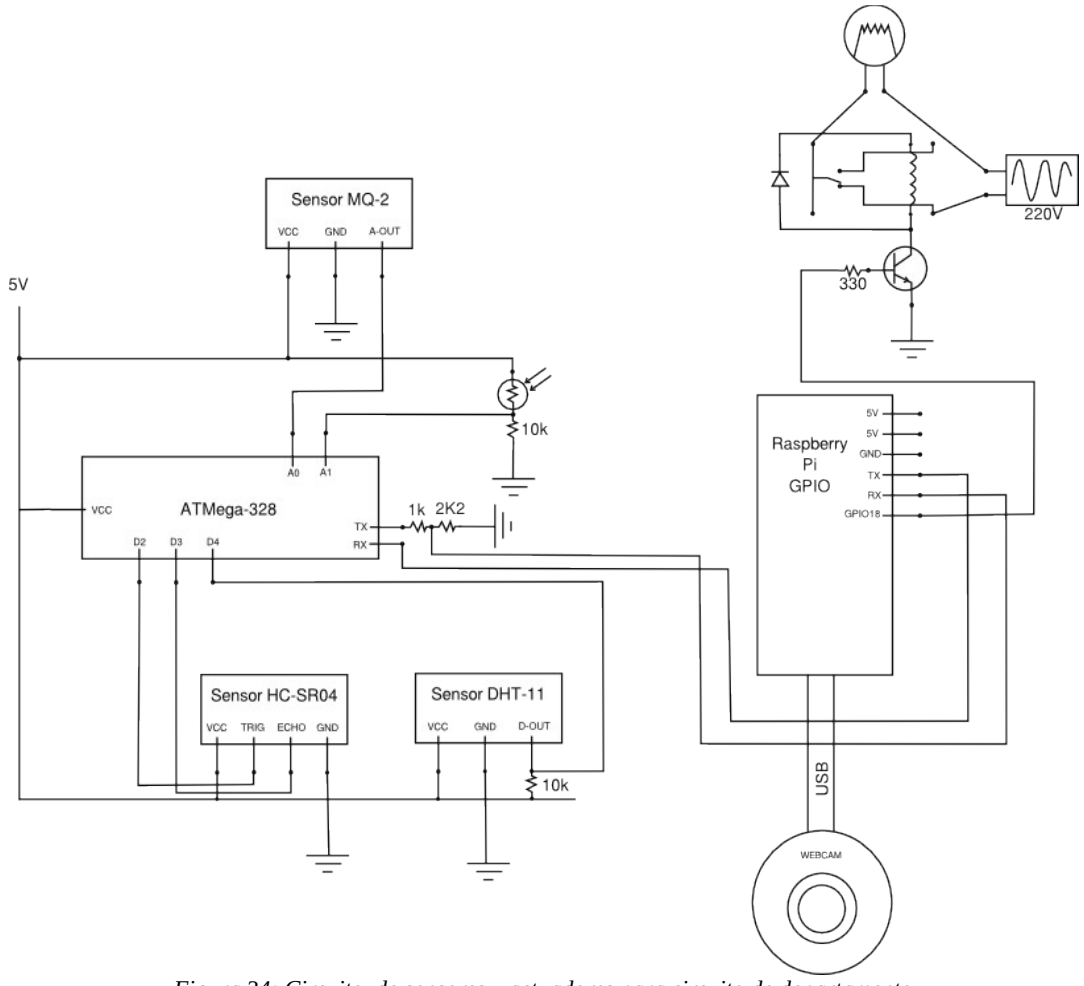

Figura 34: Circuito de sensores y actuadores para circuito de departamento

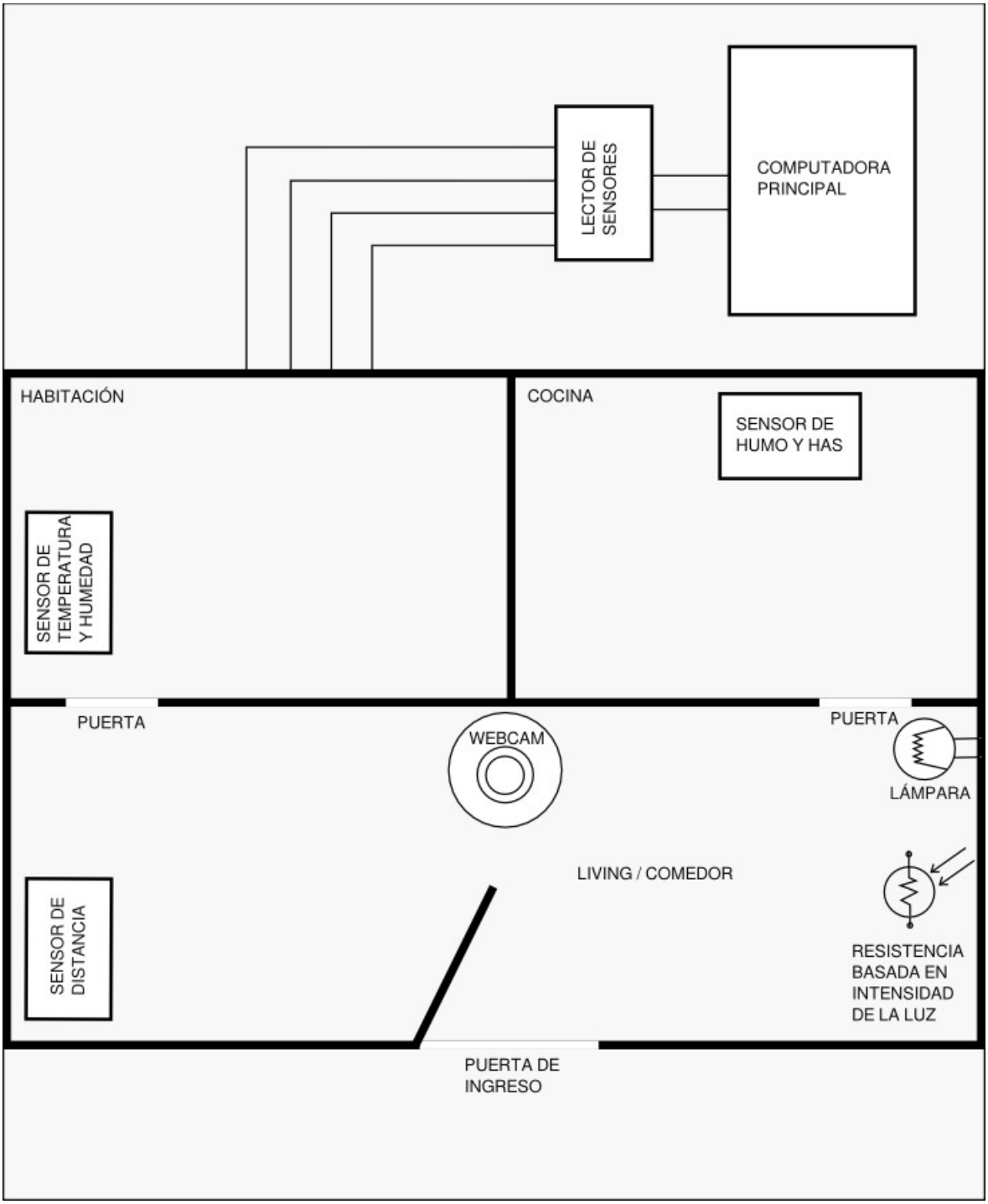

Figura 35: Distribución de la maqueta y los sensores

La Figura 36 y la Figura 37 muestran el diseño de la maqueta realizada para esta prueba.

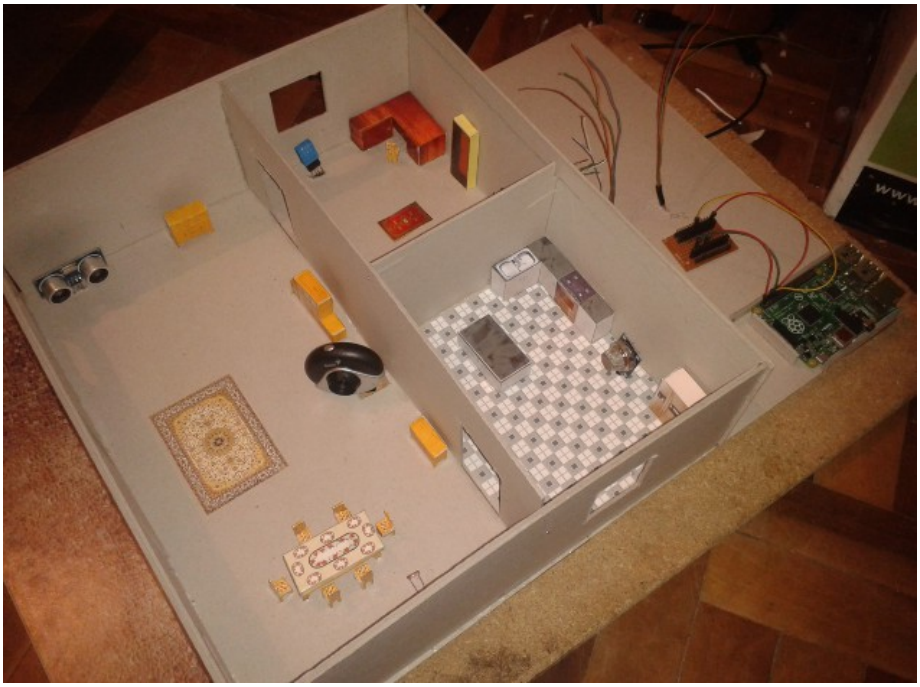

Figura 36: Diseño de maqueta para prueba domótica (1)

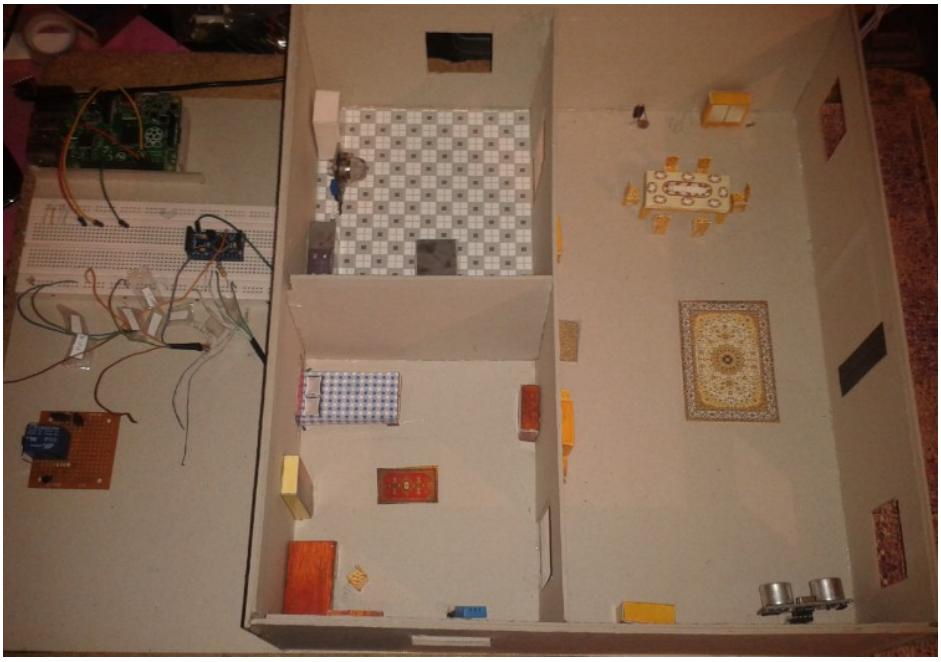

Figura 37: Diseño de maqueta para prueba domótica (2)

Resultados de la prueba

Los resultados que se obtienen en esta prueba se definen con el tiempo de respuesta tanto en sensores como actuadores. La Tabla 60 muestra los datos de respuesta con un tiempo de funcionamiento de cuatro horas con un ancho de banda promedio de 512 Kbbps.

Valor	**Resultado**
Datos enviados a servidor	14400 datos (envío cada segundo)
Tiempo de respuesta promedio entre envió y visualización en plataforma web	400 milli segundos
Peor tiempo de respuesta de actuador con caída de ancho de banda	3 segundos

Tabla 60: Resultados prueba domótica

4.1.2. Prueba en el sistema de frenado de un automóvil (Sensor/Actuador)

Esta prueba tiene como característica principal que el monitoreo y control del dispositivo a probar se encontrará en movimiento causando posibles cortes de internet o señal satelital. Junto al Universitario Gustavo Alanoca se realiza esta prueba haciendo uso de un Raspberry Pi para la programación de la lógica y un módulo SIM-5218 que permitirá a este conectarse a la plataforma Beehive a través de una red 3G además de obtener la posición del dispositivo haciendo uso del GPS que este lleva incluido.

Descripción

Esta prueba se encarga de controlar remotamente la geolocalización y el sistema de frenado de un automóvil a través de la plataforma Beehive como mecanismo de control. El usuario que tenga tenga permisos al dispositivo creado tienes las opciones de iniciar el frenado, finalizar el frenado, activar el GPS y desactivar el GPS.

La Figura 38 muestra los comandos disponibles para este dispositivo. Un usuario tras activar o desactivar el GPS puede visualizar en tiempo real las posiciones del automóvil en un mapa dentro del panel como muestra la Figura 39.

Figura 38: Comandos para el control remoto de automóvil

Figura 39: Visualización de ruta y posiciones de un automóvil

Componentes Necesarios

La Tabla 61 muestra los componentes necesarios para esta prueba.

Dispositivo	**Descripción**
Relé	Sirven como switchs para altos voltajes que pueden ser activados con pequeños pulsos
Nivelador de tensión	Al hacer uso de mucha corriente es necesario nivelarla
Módulo SIM-5218	Permite la conexión a Internet a través de una red 3G

Tabla 61: Componentes Necesarios para la prueba

Condiciones iniciales en la plataforma Beehive

Dentro de la plataforma Beehive es necesario crear un Template con los comandos/argumentos y data streams necesarios que permitan llevar a cabo las tareas del dispositivo de prueba, la Tabla 62 muestra los detalles del Template utilizado para esta prueba.

NOMBRE DE TEMPLATE: Carbrain
COMANDOS
Nombre: Iniciar Frenado **Tipo de Comando:** Int **Comando:** 1
Nombre: Finalizar Frenado **Tipo de Comando:** Int **Comando:** 2
Nombre: Activar GPS **Tipo de Comando:** String **Comando:** “G”
Nombre: Desactivar GPS **Tipo de Comando:** String **Comando:** “H”
DATA STREAMS
Nombre: Localización

Nombre de Tópico: location
Tipo de dato: Localización
Tipo de gráfica: Mapa
Unidad: Latitud, Longitud
Símbolo de unidad: lat,lng

Tabla 62: Template en Beehive para la segunda prueba

Diseño del dispositivo de Hardware

La Figura 40 muestra el esquema del circuito necesario para el sistema de frenado del dispositivo.

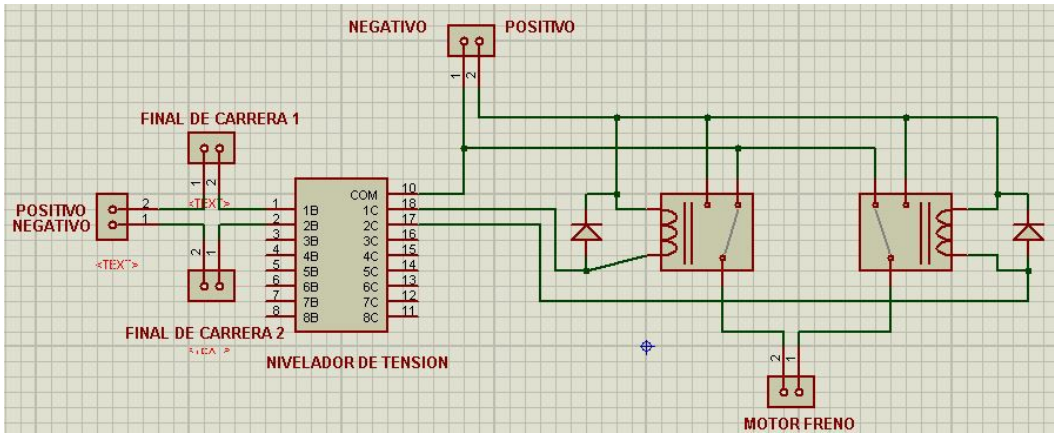

Figura 40: Circuito del sistema de frenado

La Figura 41 muestra el diseño en tercera dimensión de la circuito que será impreso en una placa.

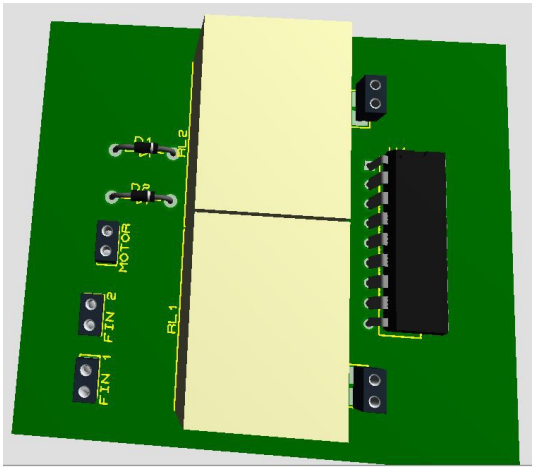

Figura 41: Diseño PCB en tercera dimensión

La Figura 42 y la Figura 43 muestran la etapa de fabricación del control del sistema de freno.

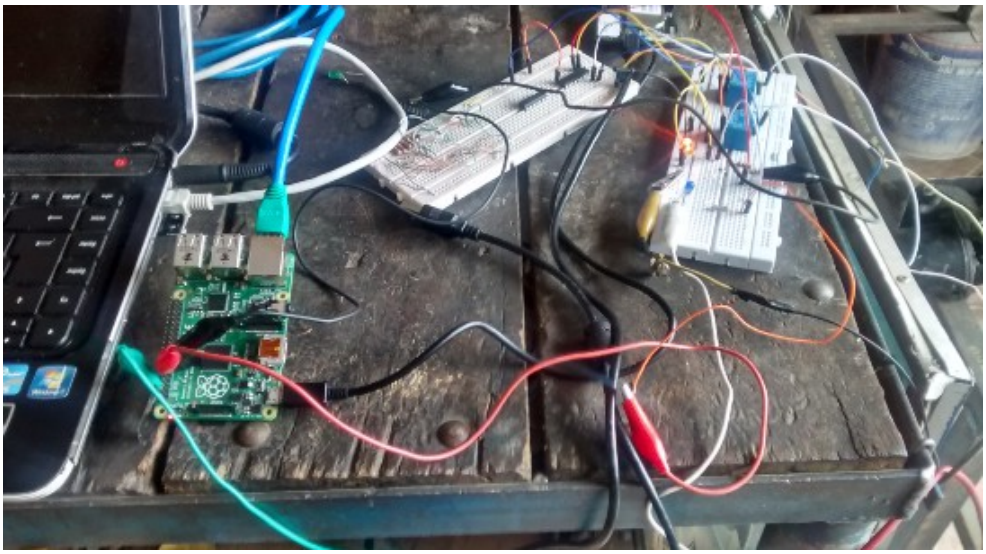

Figura 42: Circuto en placas de prueba

Figura 43: Sistema de freno del automóvil

Resultados de la prueba

Esta prueba al tener la característica principal de ser un dispositivo que se mantendrá en movimiento, se encuentran ciertos contratiempos que se escapan del control total tanto a nivel de hardware como de software. En ciertos puntos existen puntos ciegos en los que las señales telefónicas no se encuentran en el rango. Del mismo modo las condiciones climatológicas afectan al sistema GPS que requiere un enlace satelital. Tomando todos estos posibles inconvenientes, se desarrollan mecanismos de prevención en el caso de la ejecución de comandos, es decir, si la diferencia entre el tiempo de envío y el tiempo de recepción de un comando supera el minuto, este no se ejecutara, de este modo se previene la ejecución de comandos en tiempos que no se lo requiera.

La Tabla 63 muestra los tiempos de respuesta en esta prueba.

Valor	**Resultado**
Datos enviados a servidor	240 datos (durante una hora cada 15 seg.)
Tiempo de respuesta promedio entre envió y visualización en plataforma web en áreas céntricas	600 milli segundos
Peor tiempo de respuesta de actuador en lugares donde la señal no es buena	1.6 minutos

Tabla 63: Resultados prueba automóvil

4.2. Resultados de Pruebas de Estrés

Las pruebas de estrés se realizan a los diferentes recursos de la API RESTful que provee la plataforma Beehive. Estas pruebas son realizadas haciendo uso del software Apache Bench para probar los servicios RESTful.

Los elementos de prueba son dos instancias de la plataforma Beehive en un servidor local y un servidor remoto (ver sección 3.7).

4.2.1. Prueba de Estrés en los Servicios RESTful

La API RESTful que provee la plataforma Beehive (ver sección 3.4.6.) permiten acceder a los

siguientes recursos: Dispositivos, Templates, Comandos/Argumentos y Data Streams. Las pruebas de estrés a estos servicios constan de 5000 peticiones y un nivel de concurrencia de 1000, es decir, se realizarán 5000 peticiones con 1000 conexiones concurrentes. Las peticiones HTTP a realizar son del tipo de lectura (método GET) ya que esta operación es la que realiza más procesamiento en el servidor.

Recurso: Dispositivos

La Tabla 64 muestra el resultado de las pruebas en un servidor local y un servidor remoto.

Campo	Servidor Local	Servidor Remoto
Peticiones por segundo *[#/seg]*	23.43	27.28
Tiempo por petición (media) *[ms]*	42671.766	36658.560
Tiempo por petición (media en todas las peticiones concurrentes) *[ms]*	42.672	36.659
Velocidad de transferencia *[Kbyts/seg]*	39.65	41.72

Tabla 64: Prueba de estrés al recurso "Dispositivos"

El tiempo total en realizar las 5000 peticiones con un nivel de concurrencia de 1000 en el servidor local y remoto es de 213.359 y 183.293 segundos respectivamente. En el servidor local y remoto se identifica que el tiempo de respuesta comienza a demorar aproximadamente desde la petición 2250 y 4000 respectivamente.

La Figura 44 y la Figura 45 muestran el tiempo de respuesta según la cantidad de peticiones en un servidor local y remoto.

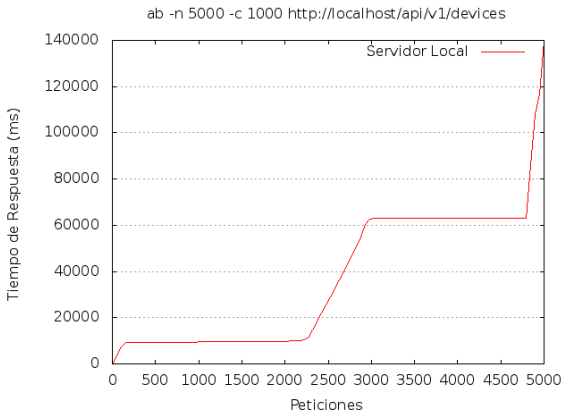

Figura 44: Recurso: Dispositivos (local)

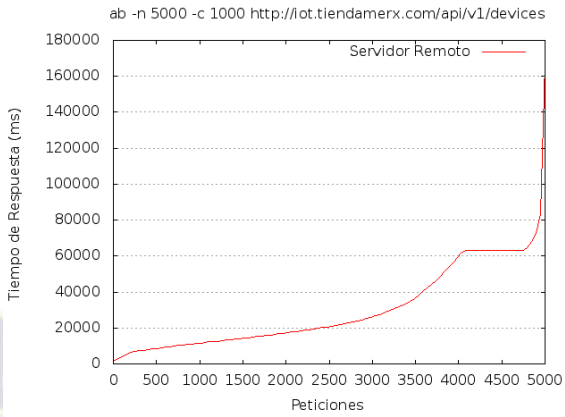

Figura 45: Recurso: Dispositivos (remoto)

Recurso: Templates

La Tabla 65 muestra el resultado de las pruebas en un servidor local y un servidor remoto.

Campo	Servidor Local	Servidor Remoto
Peticiones por segundo *[#/seg]*	21.34	27.78
Tiempo por petición (media) *[ms]*	46858.275	36003.268
Tiempo por petición (media en todas las peticiones concurrentes) *[ms]*	46.858	36.003
Velocidad de transferencia *[Kbyts/seg]*	10.28	10.77

Tabla 65: Prueba de estrés al recurso "Templates"

El tiempo total en realizar las 5000 peticiones con un nivel de concurrencia de 1000 en el servidor local y remoto es de 234.291 y 180.016 segundos respectivamente. En el servidor local y remoto se identifica que el tiempo de respuesta comienza a demorar aproximadamente desde la petición 2500 y 4000 respectivamente.

La Figura 46 y la Figura 47 muestran el tiempo de respuesta según la cantidad de peticiones en un servidor local y remoto.

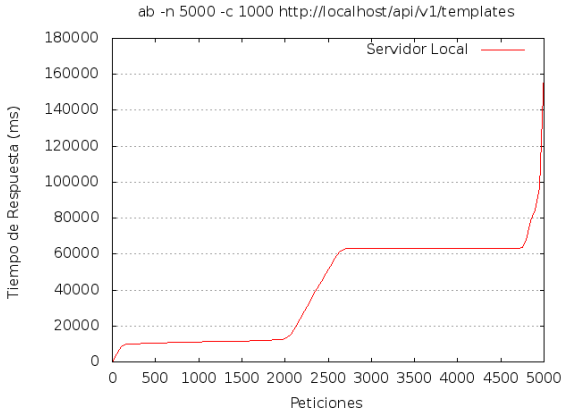

Figura 46: Recurso: Templates (local)

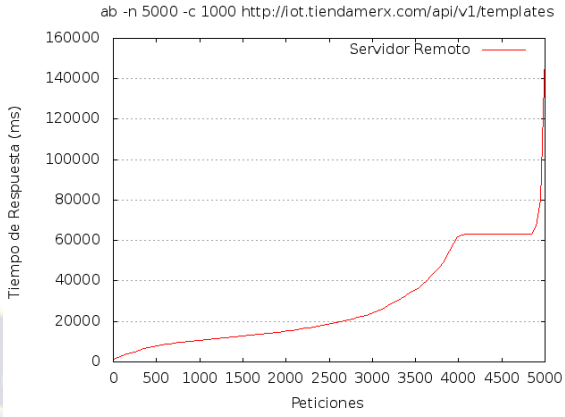

Figura 47: Recurso: Templates (remoto)

Recurso: Comandos

La Tabla 66 muestra el resultado de las pruebas en un servidor local y un servidor remoto.

Campo	Servidor Local	Servidor Remoto
Peticiones por segundo *[#/seg]*	20.20	66.57
Tiempo por petición (media) *[ms]*	49514.974	15021.932
Tiempo por petición (media en todas las peticiones concurrentes) *[ms]*	49.515	15.022
Velocidad de transferencia *[Kbyts/seg]*	4.65	37.80

Tabla 66: Prueba de estrés al recurso "Comandos"

El tiempo total en realizar las 5000 peticiones con un nivel de concurrencia de 1000 en el servidor local y remoto es de 247.575 y 75.110 segundos respectivamente. En el servidor local y remoto se identifica que el tiempo de respuesta comienza a demorar aproximadamente desde la petición 2500 y 4250 respectivamente.

La Figura 48 y la Figura 49 muestran el tiempo de respuesta según la cantidad de peticiones en un servidor local y remoto.

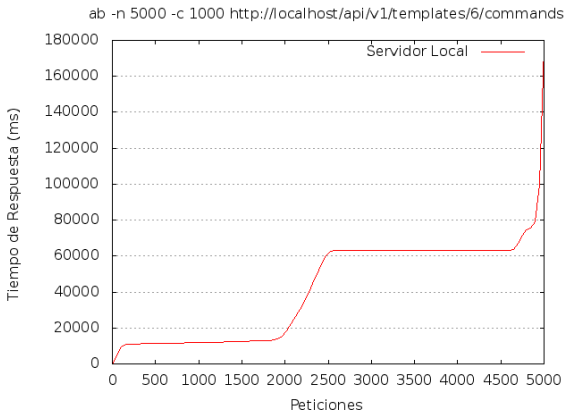

Figura 48: Recurso: Comandos (local)

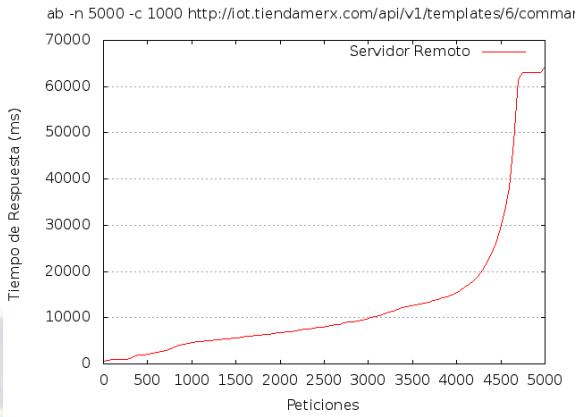

Figura 49: Recurso: Comandos (remoto)

Recurso: Data Streams

La Tabla 67 muestra el resultado de las pruebas en un servidor local y un servidor remoto.

Campo	Servidor Local	Servidor Remoto
Peticiones por segundo *[#/seg]*	20.33	66.54
Tiempo por petición (media) *[ms]*	49195.406	15027.437
Tiempo por petición (media en todas las peticiones concurrentes) *[ms]*	49.195	15.027
Velocidad de transferencia *[Kbyts/seg]*	5.52	39.44

Tabla 67: Prueba de estrés al recurso "Data Stream"

El tiempo total en realizar las 5000 peticiones con un nivel de concurrencia de 1000 en el servidor local y remoto es de 245.977 y 75.137 segundos respectivamente. En el servidor local y remoto se identifica que el tiempo de respuesta comienza a demorar aproximadamente desde la petición 2500 y 4250 respectivamente.

La Figura 50 y la Figura 51 muestran el tiempo de respuesta según la cantidad de peticiones en un servidor local y remoto.

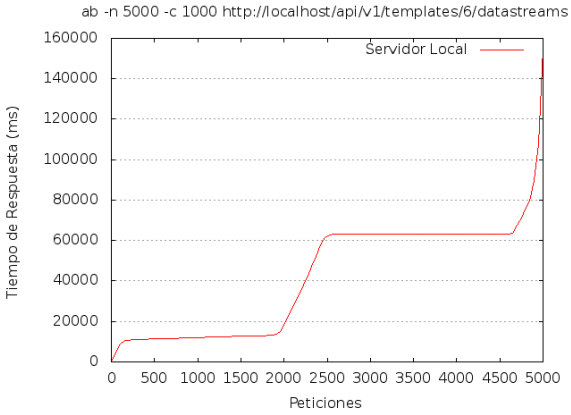

Figura 50: Recurso: Data Stream (local)

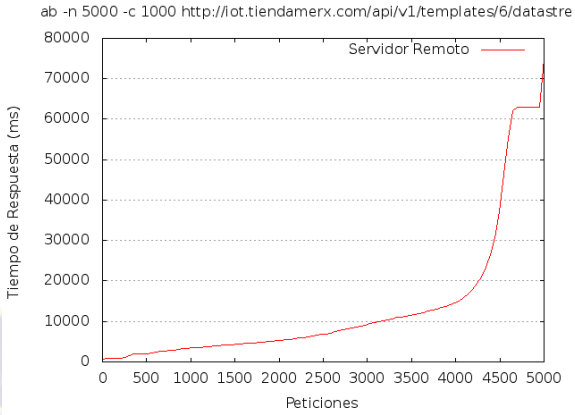

Figura 51: Recurso: Data Stream (remoto)

Es notorio que los comportamientos son similares para cada recurso.

Los datos de tiempo de conexión, tiempo de procesamiento, tiempo de espera y tiempo total fueron generados por el software Apache Bench, estos archivos se encuentran disponibles en la siguiente dirección: https://gist.github.com/donkeysharp.

CAPÍTULO 5
CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

- Se cumple el objetivo general al haber desarrollado una plataforma Open-Source para el Internet De Las Cosas, la cual contiene las características de gestión de usuarios, gestión de permisos sobre dispositivos, librerías para dispositivos de hardware (Arduino y Raspberry Pi B+) y una API RESTful para el uso de la plataforma por otros desarrolladores.
- La estructura de la plataforma Beehive al hacer uso de principios de diseño S.O.L.I.D. permite que el código de la plataforma sea mantenible y extendible a nivel de código.
- La reusabilidad por parte de otros desarrolladores de distintas arquitecturas y plataformas es posible gracias a la API RESTful que la plataforma provee.
- La estructura de la plataforma Beehive se plantea de la forma más personalizable posible, es decir, el usuario puede personalizar los canales de comunicación de los dispositivos como este lo requiera y así poder visualizar los datos que estos generen.
- Se logran crear librerías de hardware para Arduino y Raspberry Pi B+ que permiten la comunicación con la plataforma Beehive (ver capítulo 4).

5.2. Recomendaciones

- A medida que la cantidad de dispositivos de hardware a ser conectados crece, es necesario que las características de hardware del servidor (en el que la instancia de la plataforma Beehive se ejecuta) sean aumentadas (mayor memoria RAM, almacenamiento, procesamiento y otros).
- Para poder ser utilizada en diferentes lenguajes se recomienda hacer la implementación de internacionalización (I18N) lo cual permitirá que la plataforma pueda ser visualiza-

da en diferentes lenguajes

- Es necesario realizar la compilación y distribución de la librería de autenticación para Mosquitto para las diferentes arquitecturas y distribuciones de GNU/Linux
- Para propósitos de escalabilidad, es necesario probar diferentes instancias de Mosquitto en un cluster de servidores.
- Actualmente las librerías de hardware se limitan para dispositivos Arduino y Raspberry Pi B+, por lo cual, es necesario implementar las librerías para otros dispositivos de hardware como micro controladores PIC.

5.3. Recomendaciones para futuros trabajos

- Para que un dispositivo pueda relacionarse con otros dispositivos de hardware es necesario que la plataforma cuente con un mecanismo para Procesamiento Complejo de Eventos.
- Hacer un análisis de las datos generados por la plataforma haciendo uso de técnicas como Map Reduce, Inteligencia Artificial y otros.

BIBLIOGRAFÍA

[ARDUINO: 2014], Arduino, Arduino Concept, http://arduino.cc, [Acceso: noviembre 2014]

[ARDUINO-UNO: 2014], Arduino Uno, Arduino Uno Overview, http://arduino.cc/en/Main/ArduinoBoardUno, [Acceso: noviembre 2014]

[ASAY: 2014] – Asay Alan, 2014, The Internet Of Things Will Need Millions Of Developers By 2020, http://readwrite.com/2014/06/27/internet-of-things-developers-jobs-opportunity [Acceso: noviembre 2014]

[ATMEGA: 2014], Atmel, 2014, ATMEGA328 Specifications, http://www.atmel.com/devices/atmega328.aspx, [Acceso: noviembre 2014]

[BECK y OTROS: 2001], Beck Kent; Beedle Mike; Bennekum Arie; Cockburn Alistar; Cunningham Ward; Fowler Martin; Grenning James; Highsmith Jim; Hunt Adrew; Jeffries Ron; Kern Jon; Marick Brian; Martin Robert; Mellor Steve; Schwaber Ken; Sutherland Jeff; Thomas Dave, 2001, Principios del Manifiesto Ágil, http://agilemanifesto.org/iso/es/principles.html, [Acceso: abril 2015]

[CC3000: 2012], CC3000 - Texas Instruments, 2012, CC3000 Datasheet, http://www.ti.com/lit/ds/symlink/cc3000.pdf, [Acceso: abril 2015]

[DACOSTA:2013] – Francis daCosta, 2013, Rethinking The Internet Of Things, A Scalable Approach to Connecting Everything, Appress Open

[DIXON: 2007], Dixon James, 2007, Open Scrum, http://wiki.pentaho.com/display/OpenScrum, [Acceso: febrero 2015]

[ETHERNETSHIELD: 2013], Arduino Ethernet Shield, 2013, Arduino Ethernet Shield, http://arduino.cc/en/Main/ArduinoEthernetShield, [Acceso: abril 2015]

[FIELDING: 2000], Fielding Roy Thomas, 2000, Architectural Styles and Design of Network-based Software Architectures (Capítulo 5 – Representational State Transfer REST), http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, [Acceso: noviembre 2014]

[GARTNER: 2013] – Gartner, 2013, Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020, http://www.gartner.com/newsroom/id/2636073 [Acceso: noviembre 2014]

[GOOGLE-NEST: 2014] – Nest Labs, 2014, Google to Acquire Nest,

https://investor.google.com/releases/2014/0113.html [Acceso: octubre 2014]

[GUTIERREZ: 2014] – Gutierrez Luis, 2014, Sharing Robots http://sourceforge.net/projects/sharingrobots/ [Acceso: Octubre 2014]

[HELCEN: 2012] Helsen Bruce, 2012, Internet Controlled Desk http://www.instructables.com/id/Internet-Controlled-Desk-Lamp/ [Acceso: Julio 2014]

[HIVEMQ: 2015], HiveMQ, 2015, MQTT Essentials Part2, Publish & Subscribe, http://www.hivemq.com/mqtt-essentials-part2-publish-subscribe/, [Acceso: abril 2015]

[HÖLLER: 2014] - J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, D. Boyle, 2014, From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence, Editorial: Elsevier

[HTTP: 1999] - RFC26126, 1999, Hypertext Transfer Protocol -- HTTP/1.1 (Section 9 - Method Definitions), http://tools.ietf.org/html/rfc2616#section-9, [Acceso: noviembre 2014]

[ISAACS: 2014] – Isaacs Charlie, Forbes, 2014, 3 Ways The Internet Of Things Is Revolutionizing Health Care, http://www.forbes.com/sites/salesforce/2014/09/03/nternet-things-revolutionizing-health-care/, [Acceso: marzo 2015]

[JASPER: 2014] – Jasper Technologies, 2014, Connected Car Cloud, http://www.jasper.com/products/connected-car-cloud [Acceso: octubre 2014]

[KOROLOV: 2015] – Korolov Maria, Network World, 2015, Will open source save the Internet of Things?, http://www.networkworld.com/article/2902231/internet-of-things/will-open-source-save-the-internet-of-things.html, [Acceso: marzo 2015]

[LEGGETTER: 2013] – Phil Leggetter, 2013, Choosing Your Realtime Web App Tech Stack, http://www.leggetter.co.uk/2013/12/09/choosing-realtime-web-app-tech-stack.html [Acceso: octubre 2014]

[LOCKITRON: 2014], Lockitron, Lockitron, http://lockitron.com, [Acceso: octubre 2014]

[MARTIN&MARTIN: 2007], Martin Robert C. & Martin Micah, 2007, Agile Principles, Patterns and Practices in C#, Prentice Hall.

[MCEWEN, CASSIMALLY: 2014] – Adrian McEwen, Hakim Cassimally, 2014, Designing The Internet Of Things, John Wiley and Sons, Ltd.

[MESHBLU: 2014], Meshblu, Meshblu, https://github.com/octoblu/meshblu, [Acceso: octubre 2014]

[MOHAMMED: 2014], Jahangir Mohammed, 2014, Surprise: Agriculture is doing more with IoT Innovation than most other industries, http://venturebeat.com/2014/12/07/surprise-agriculture-is-doing-more-with-iot-innovation-than-most-other-industries/, [Acceso marzo 2015]

[MOOREHEAD: 2013], Moorheade Patrik, Forbes, 2013, The Problem With Home Automation's Internet Of Things (IoT), http://www.forbes.com/sites/patrickmoorhead/2013/09/26/the-problem-with-home-automations-iot/, [Acceso: marzo 2015]

[MQTT: 2004], MQTT Protocol, Protocol Definition, http://mqtt.org/ [Acceso: noviembre 2014]

[MQTT-OASIS: 2014], MQTT & Oasis, 2014, MQTT Version 3.1.1 Specification, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718018, [Acceso: abril 2015]

[OTWELL: 2013], Otwell Taylor, 2013, Laravel: From Apprentice To Artisan – Advanced Architecture With Laravel 4, Leanpub

[RASPBERRYPI: 2012], Raspberry Pi, 2012, What is a Raspberry Pi?, https://www.raspberrypi.org/help/what-is-a-raspberry-pi/, [Acceso: marzo 2015]

[RASPBERRYPIBPLUS: 2014], Raspberry Pi B+, 2014, Introducing Raspberry Pi model B+, https://learn.adafruit.com/downloads/pdf/introducing-the-raspberry-pi-model-b-plus-plus-differences-vs-model-b.pdf, [Acceso: marzo 2015]

[RAYMOND: 2000], Raymod Eric, 2000, The Cathedral and Bazaar, http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/, [Acceso: abril 2015]

[RICHARDSON, RUBY: 2007], Richardson Leonard & Ruby Sam, 2007, RESTful Web Services, O'Relly Media

[SCHUERMANS: 2014], Stijin Schuermans, 2014, The Kingmakers of the Internet Of Things, http://www.visionmobile.com/blog/2014/10/kingmakers-internet-things, [Acceso: marzo 2015]

[SCHWABER&SUTHERLAND: 2013], Schwaber Ken, Sutherland Jeff, 2013, The Scrum

Guide http://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf, [Acceso: abril 2015]

[SIM5218: 2013], Shield SIM5218, Cooking Hacks, 2013, 3G + GPS Shield for Arduino, http://www.cooking-hacks.com/documentation/tutorials/arduino-3g-gprs-gsm-gps, [Acceso: abril 2015]

[SIM5218-AT: 2011] – SIMCom, 2011, SIM5218 Serial AT Command Manual V1.21, http://www.cooking-hacks.com/skin/frontend/default/cooking/pdf/SIM5218_AT_command_manual.pdf, [Acceso: abril 2015]

[THINGSPEAK: 2014], Thing Speak, Thing Speak, 2014, https://github.com/iobridge/thingspeak, [Acceso: octubre 2014]

[VCPOST: 2014] – Venture Capital Post, 2014, California firm Jasper lands $50M in funding to bring the Internet of Things to Coca-Cola, http://www.vcpost.com/articles/23402/20140417/california-firm-jasper-lands-50m-in-funding-to-bring-the-internet-of-things-to-coca-cola.htm [Acceso: octubre 2014]

[WEBSOCKET: 2011], WebSocket Protocol, 2011, The WebSocket Protocol, Official RFC (Request For Comments), http://tools.ietf.org/html/rfc6455, [Acceso: noviembre 2014]

[XIVELY: 2011], Xively, 2011, Xively By LogMeIn, http://xively.com, [Acceso: octubre 2014]

[ZHANG: 2011], Zhang Lucy, 2011, Building Facebook Messenger, https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920 [Acceso: abril 2015]

ANEXOS

Anexo A – Estadísticas de proyectos Open-Source "Internet Of Things"

En la actualidad, GitHub (www.github.com) es el repositorio de proyectos Open-Source más grande del mundo. En este se encuentran los proyectos Open-Source más populares e importantes como el Kernel Linux, Spring Framework, Proyecto Arduino entre otros.

Tomando en cuenta los proyectos que se encuentran en GitHub se desea mostrar que la cantidad de proyectos Open-Source relacionadas al Internet De Las Cosas es baja en comparación a otro tipo de proyectos en GitHub como ser Frameworks Web o Aplicaciones Móviles.

Considerando un subconjunto de proyectos de GitHub de 13,290 y 14,461 (septiembre y octubre del 2014 respectivamente), el 2.39% y 3.31% se encuentra relacionado al Internet De Las Cosas. En las siguiente tablas se puede ver claramente que la cantidad de proyectos relacionados al Internet De Las Cosas es baja comparada a otro tipo de proyectos:

Nro. Proyectos Internet De Las Cosas	Nro. Proyectos Frameworks Web	Nro. Proyectos Aplicaciones Móviles
317	8,647	4,326

Información recolectada el 3 de septiembre del 2014

Nro. Proyectos Internet De Las Cosas	Nro. Proyectos Frameworks Web	Nro. Proyectos Aplicaciones Móviles
478	9,199	4,784

Información recolectada el 22 de octubre del 2014

GitHub usa las siguientes características de un proyecto para medir su popularidad:

- "Forks" - indica el número de desarrolladores que probablemente sen encuentren interesados en aportar a un proyecto
- "Stars" - indica el número de desarrolladores que se encuentran atentos a los cambios que se realizan en un proyecto

Entre los 478 repositorios relacionados al Internet De Las Cosas (octubre 2014), solo 8 muestran un interés superior a 50 tanto en “forks” o “stars”.

Dentro de los primeros 20 repositorios que muestran más interés, es posible categorizarlos de la siguiente forma:

Repositorio	**Categoría**	**Forks**	**Stars**	**Última contribución**
Contiki	Sistema operativo para microcontroladores	273	1122	Octubre 2014
Node-red	Herramienta visual para conexión y control de flujos entre dispositivos	205	920	Octubre 2014
Meshblu	Plataforma de mensajería y comunicación máquina-máquina	110	519	Octubre 2014
Thingspeak	Plataforma de control sobre el internet de las cosas	68	330	Octubre 2014
Exciting-IO/Printer	Plataforma para manejo remoto de impresoras	58	230	Julio 2014
Steward	Plataforma de control sobre el internet de las cosas	53	221	Agosto 2014
OpenIoT	Middleware para el procesamiento de información del internet de las cosas	38	79	Octubre 2014
YeeLinkLib	Librería para microcontroladores ATMega	21	44	Junio 2013
Things-api	Conjunto de librerías para el internet de las cosas	17	34	Marzo 2014
HomeGenie	Servidor para automatización del hogar	22	18	Octubre 2014
Qest	Integración protocolo MQTT y REST	21	91	Abril 2013
Android-	Adaptador para conectar android	12	19	Junio 2014

Repositorio	Categoría	Forks	Stars	Última contribución
BleEventAdapter	con bluetooth			
UioT	Micro Gateway para dispositivos	6	5	Septiembre 2013
Internet-of-things	Fragmentos de código para el módulo ethernet de arduino	4	0	Julio 2014
Sensemote	Sistema operativo para microcontroladores	4	12	Octubre 2012
Fractalide	Herramienta para control de flujo	4	6	Julio 2014
Iot-assistant	Aplicación web para impresoras Adafruit	4	20	Abril 2013
2014IOT	Documentos sobre un curso digital sobre el internet de las cosas	4	1	Agosto 2014
IoT-Buddy	Aplicación iOS para control de dispositivos	4	4	Julio 2013
Iot	Proyecto para arduino y raspberry pi (chino)	35	89	Octubre 2014

Conclusión

Analizando estos datos, se puede concluir que la cantidad de proyectos Open-Source relacionados al Internet De Las Cosas comparada con otro tipo de proyectos es baja, pero no se puede dejar por detrás que la cantidad de proyectos creció en más de 100 en aproximadamente dos meses, lo cual indica que el Internet De Las Cosas esta captando más interés en los desarrolladores.

Por otro lado, al ver la última fecha de contribución de los 20 proyectos más populares, 8 de estos mantiene un desarrollo constante hasta la fecha (octubre 2014), a diferencia de los demás cuyo desarrollo se encuentra pausado por más de un mes incluso años en el caso de "Sensemote"

ANEXO B – Documentación de la API RESTful

Nombre: Obtener token de acceso
Recurso: Ninguno
Endpoint: /api/v1/auth
Method: POST

Payload:
{
“username”: <usuario>
“password”: <password>
}

Resultado:
{
“token” : <token de acceso>
}

Nombre: Obtener lista de templates
Recurso: Template
Endpoint: /api/v1/templates?token=<token de acceso>
Method: GET
Payload: Ninguno
Resultado: [{ “id”: <id>, “name”: <name>, “description”: <description> }, { “id”: <id>, “name”: <name>, “description”: <description> }, ...]

Nombre: Obtener un template específico
Recurso: Template
Endpoint: /api/v1/templates/<id_template>?token=<token de acceso>
Method: GET
Payload: `Ninguno`

Resultado:
`{`
`  “id”: <id_template>,`
`  “name”: <name>,`
`  “description”: <description>`
`}`

Nombre: Crear un template
Recurso: Template
Endpoint: /api/v1/templates?token=<token de acceso>
Method: POST

Payload:
`{`
`  “name”: <name>,`
`  “description”: <description>`
`}`

Resultado:
`{`
`  “id”: <id>,`
`  “name”: <name>,`
`  “description”: <description>`
`}`

Nombre: Actualizar un template
Recurso: Template
Endpoint: /api/v1/templates/<id_template>?token=<token de acceso>

Method: PUT

Payload:
{
“name”: <name>,
“description”: <description>
}

Resultado:
{
“id”: <id_template>,
“name”: <name>,
“description”: <description>
}

Nombre: Obtener lista de comandos
Recurso: Comando
Endpoint: /api/v1/templates/<template_id>/commands?token=<token de acceso>
Method: GET
Payload: Ninguno
Resultado: [{ “id”: <id>, “name”: <name>, “short_cmd”: <short command>, “cmd_type”: <command type>, “template_id”: <template_id>, “arguments”: [“id”: <argument_id>, “name”: <argument_name>, “type”: <argument_type>, “default”: <default_value>, “minimum”: <minimum_value>, “maximum”: <maximum_value>] }, ...

```
]
```

Nombre: Obtener un comando específico

Recurso: Comando

Endpoint: /api/v1/templates/<id_template>/commands/<id_command>?token=<token de acceso>

Method: GET

Payload:
Ninguno

Resultado:

```
{
    “id”: <id_command>,
    “name”: <name>,
    “short_cmd”: <short command>,
    “cmd_type”: <command type>,
    “template_id”: <template_id>,
    “arguments”: [
      “id”: <argument_id>,
      “name”: <argument_name>,
      “type”: <argument_type>,
      “default”: <default_value>,
      “minimum”: <minimum_value>,
      “maximum”: <maximum_value>
    ]
  }
```

Nombre: Crear un comando

Recurso: Comando

Endpoint: /api/v1/templates/<id_template>/commands?token=<token de acceso>

Method: POST

Payload:

```
{
    “name”: <name>,
    “short_cmd”: <short command>,
    “cmd_type”: <command type>
```

}
Resultado:
{
“id”: <id>,
“name”: <name>,
“short_cmd”: <short command>,
“cmd_type”: <command type>
}

Nombre: Actualizar un template
Recurso: Comando
Endpoint: /api/v1/templates/<id_template>/commands/<id_command>?token=<token de acceso>
Method: PUT

Payload:
{
“name”: <name>,
“short_cmd”: <short command>,
“cmd_type”: <command type>
}

Resultado:
{
“id”: <id>,
“name”: <name>,
“short_cmd”: <short command>,
“cmd_type”: <command type>
}

Nombre: Obtener lista de data streams
Recurso: Data Stream
Endpoint: /api/v1/templates/<template_id>/datastreams?token=<token de acceso>
Method: GET
Payload: Ninguno

Resultado:

```
[
  {
    “id”: <id>,
    “name”: <name>,
    “topic_name”: <topic_name>,
    “data_type”: <data type>,
    “unit”: <unit>,
    “unit_symbol”: <unit_symbol>,
    “display_type”: <display_type>,
    “template_id”: <template_id>
  },
  ...
]
```

Nombre: Obtener un data stream específico
Recurso: Data Stream
Endpoint: /api/v1/templates/<id_template>/datastreams/<id_datastream>?token=<token de acceso>
Method: GET
Payload: `Ninguno`

Resultado:
`{`
`  “id”: <id_datastream>,`
`  “name”: <name>,`
`  “topic_name”: <topic_name>,`
`  “data_type”: <data type>,`
`  “unit”: <unit>,`
`  “unit_symbol”: <unit_symbol>,`
`  “display_type”: <display_type>,`
`  “template_id”: <template_id>`
`}`

Nombre: Crear un data stream
Recurso: Data Stream

Endpoint: /api/v1/templates/<id_template>/datastreams?token=<token de acceso>

Method: POST

Payload:

```
{
    “name”: <name>,
    “topic_name”: <topic_name>,
    “data_type”: <data type>,
    “unit”: <unit>,
    “unit_symbol”: <unit_symbol>,
    “display_type”: <display_type>
}
```

Resultado:

```
{
    “id”: <id>,
    “name”: <name>,
    “topic_name”: <topic_name>,
    “data_type”: <data type>,
    “unit”: <unit>,
    “unit_symbol”: <unit_symbol>,
    “display_type”: <display_type>,
    “template_id”: <template_id>
}
```

Nombre: Actualizar un data stream

Recurso: Data Stream

Endpoint: /api/v1/templates/<id_template>/datastreams/<id_datastream>?token=<token de acceso>

Method: PUT

Payload:

```
{
    “name”: <name>,
    “topic_name”: <topic_name>,
    “data_type”: <data type>,
    “unit”: <unit>,
    “unit_symbol”: <unit_symbol>,
    “display_type”: <display_type>
}
```

Resultado:

```
{
    “id”: <id_datastream>,
    “name”: <name>,
    “topic_name”: <topic_name>,
    “data_type”: <data type>,
    “unit”: <unit>,
    “unit_symbol”: <unit_symbol>,
    “display_type”: <display_type>,
    “template_id”: <template_id>
}
```

Nombre: Obtener lista de dispositivos
Recurso: Dispositivo
Endpoint: /api/v1/devices?token=<token de acceso>
Method: GET
Payload: `Ninguno`

Resultado:
`[`
`  {`
`    “id”: <id>,`
`    “name”: <name>,`
`    “description”: <description>,`
`    “template_id”: <template_id>,`
`    “serial_number”: <serial_number>,`
`    “device_secret”: <device_secret>,`
`    “pub_key”: <publish_key>,`
`    “sub_key”: <subscribe_key>`
`  },`
`  ...`
`]`

Nombre: Obtener un dispositivo específico
Recurso: Dispositivo

Endpoint: /api/v1/devices/<id_device>?token=<token de acceso>
Method: GET
Payload: Ninguno

Resultado:
{
“id”: <id_device>,
“name”: <name>,
“description”: <description>,
“template_id”: <template_id>,
“serial_number”: <serial_number>,
“device_secret”: <device_secret>,
“pub_key”: <publish_key>,
“sub_key”: <subscribe_key>
}

Nombre: Crear un dispositivo
Recurso: Dispositivo
Endpoint: /api/v1/devices?token=<token de acceso>
Method: POST

Payload:
{
“name”: <name>,
“description”: <description>,
“template_id”: <template_id>
}

Resultado: { “id”: <id>, “name”: <name>, “description”: <description>, “template_id”: <template_id>, “serial_number”: <serial_number>, “device_secret”: <device_secret>, “pub_key”: <publish_key>, “sub_key”: <subscribe_key>

```
}
```

Nombre: Actualizar un dispositivo
Recurso: Dispositivo
Endpoint: /api/v1/devices/<id_device>?token=<token de acceso>
Method: PUT

Payload:
{
“name”: <name>,
“description”: <description>
}

Resultado:
{
“id”: <id_device>,
“name”: <name>,
“description”: <description>,
“template_id”: <template_id>,
“serial_number”: <serial_number>,
“device_secret”: <device_secret>,
“pub_key”: <publish_key>,
“sub_key”: <subscribe_key>
}